Truthful and Fair Resource Allocation

Truthful and Fair Resource Allocation

A dissertation presented
by

John Kwang Lai

to
The School of Engineering and Applied Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of

Computer Science

Harvard University
Cambridge, Massachusetts
April 2013

Thesis advisor
David C. Parkes

Author
John Kwang Lai

Truthful and Fair Resource Allocation

Abstract

How should we divide a good or set of goods among a set of agents? There are various constraints that we can consider. We consider two particular constraints. The first is fairness – how can we find fair allocations? The second is truthfulness – what if we do not know agents valuations for the goods being allocated? What if these valuations need to be elicited, and agents will misreport their valuations if it is beneficial? Can we design procedures that elicit agents' true valuations while preserving the quality of the allocation?

We consider truthful and fair resource allocation procedures through a computational lens. We first study fair division of a heterogeneous, divisible good, colloquially known as the cake cutting problem. We depart from the existing literature and assume that agents have restricted valuations that can be succinctly communicated. We consider the problems of welfare-maximization, expressiveness, and truthfulness in cake cutting under this model.

In the second part of this dissertation we consider truthfulness in settings where payments can be used to incentivize agents to truthfully reveal their private information. A *mechanism* asks agents to report their private preference information and computes an allocation and payments based on these reports. The *mechanism design* problem is to find *incentive compatible* mechanisms which incentivize agents to truthfully reveal their private information and simultaneously compute allocations with desirable properties. The traditional approach to mechanism design specifies mechanisms by hand and proves that certain desirable properties are satisfied. This limits the design space to mechanisms that can be written down and analyzed. We take a computational approach, giving computational procedures that produce mechanisms with desirable properties. Our first contribution designs a procedure that modifies heuristic branch and bound search and makes it usable as the allocation algorithm in an incentive compatible mechanism. Our second contribution draws a novel connection between incentive compatible mechanisms and machine learning. We use this connection to learn payment rules to pair with provided allocation rules. Our payment rules are not exactly incentive compatibility, but they minimize a measure of how much agents can gain by misreporting.

Contents

Acknowledgements

I could not have completed this dissertation without the guidance and support of many people. First and foremost, thank you to my advisor David Parkes. David, you are the best advisor anyone could ask for and a true inspiration. Thank you for all of the brilliant research insights and guidance. Thank you for believing in me, advocating on my behalf, and being caring, supportive, and understanding throughout my journey. I would like to thank Ariel Procaccia, who introduced me to cake cutting and has been an incredible collaborator, and the other members of my committee Yiling Chen and Vince Conitzer for their time and valuable feedback. I am grateful for the EconCS group at Harvard; I could not have asked for a more nurturing environment. Specific thanks to Michal Feldman, for our collaborations and her support, to Sven Seuken and Haoqi Zhang, who stayed up late at conferences to give feedback on my talks, and to Andrew Mao, who has been a great friend and always willing to help. I've been fortunate to collaborate with many talented researchers including Moritz Bächer, Steven Brams, Ioannis Caragiannis, Tanmoy Chakraborty, Yuga Cohler, Paul Dütting, Felix Fischer, Pitchayut Jirapinyo, Ian Kash, David Kurokawa, Ben Lubin, Jamie Morgenstern, and Aviv Zohar. I was generously supported by a National Defense Science and Engineering Graduate Fellowship during my time in graduate school.

Outside of research, I have been blessed with a supportive community of friends and family. Adi, Dennis, Ian, Matt, Naimish, Nick, and Steve, thanks for your unwavering friendship and support. James and Jerry, thanks for the brotherly love. Mom and Dad, thanks for all the sacrifices you've made and still make on our behalf. Your dedication, love, and care are amazing. Wendy, Chris, and his little brother, I love you. Thank you for the joy you bring to my life.

Chapter 1

Introduction

Resource allocation problems arise in all facets of life. Individuals and families decide how to allocate their monetary resources and how to spend their time. Companies decide how to allocate their human resources across different departments and priorities. Governments must decide how to allocate military resources and how to divide pollution rights. These are a just a tiny fraction of the resource allocation problems that are being considered and solved on a daily basis.

In addition to looking at resource allocation problems across larger and larger cross-sections of society, resource allocation problems can also be classified in terms of the resources being allocated. The resources can be physical goods, such as precious metals or a painting. They can be vanishing digital goods such as advertising slots on the Internet [Edelman et al., 2007] and television [Nisan, 2010] or computing resources in the cloud. As in the examples from the previous paragraph, it is also possible to consider monetary resources as well as human capital.

The main take away is that resource allocation problems are widespread and important. More formally, in a resource allocation problem, there is some resource or set of resources that needs to be allocated among a set of agents. The agents have preferences for different allocations of the resources. A resource allocation algorithm takes the agents' preferences and determines an allocation of the resources to the agents and optionally an amount that each agent needs to pay in exchange for its allocation.

In this dissertation, we undertake the rigorous study of resource allocation algorithms with a focus on two desirable properties. The first desirable property is *fairness*, which requires that the algorithm find an allocation (and optionally payments) that is deemed fair. Fairness is a natural criterion to consider, and in certain settings fairness is an important

consideration. For instance, when the government allocates resources, it has a strong interest in fairness and not favoring any parties over others. In settings like divorce settlement, the goal is to try and give the interested parties their fair share. If people have pooled together resources for a shared resource such as computing servers, then it is important to allocate time on the server in a fair way. There are several possible definitions of fairness and we discuss the alternatives in the sequel, but an example of a natural fairness requirement is *envy-freeness*, which requires that each agent prefer its own situation to that of any other agent.

The second desirable property is *truthfulness*. While in some settings it is reasonable to assume that the true preferences of the agents are known, in many settings, agents may be self-interested and will misreport their preferences in order to increase their utility from participation. In almost all commercial allocation settings the participants are companies that are trying to maximize their profit and may or may not reveal their true preferences depending on whether it is in their best interest. For instance, in the FCC's wireless spectrum auctions, participants have used their bids to send signals to and implicitly collude with other participants [Cramton and Schwartz, 2000]. With the coming of the digital age and the Internet, many new resource allocation problems have been created, almost all of which involve self-interested agents with private information. Perhaps the best-studied example is the setting of sponsored search auctions, which provide the main source of Google's revenue [Edelman et al., 2007].

Informally, truthfulness requires that it is (weakly) in an agent's best interest to report its true preferences. Using the simple single-item auction setting as an example, consider a first-price auction and a second-price auction. Both auctions give the item to the highest bidder, but a first-price auction charges the bidder its bid while a second-price auction charges the bidder the second highest bid. The first-price auction is not truthful, since a bidder has an incentive to shade its bid. It is an easy exercise to show that an agent's best strategy is to bid its value for the item in a second-price auction, and the intuition is that the agent cannot affect its payment since it is set by the second highest bid.

One might see the appeal of truthfulness, but still wonder why truthfulness matters if non-truthful mechanisms still result in good allocations of our resources. We discuss this more in the sequel, but there are several justifications, ranging from practical to theoretical. On the practical side, truthfulness makes it easy for agents to participate and makes for procedures that are more fair in the sense that there is no advantage to being more sophisticated or having more information about othe agents. On the theoretical side, economic

theory establishes that it is in some sense without loss to study truthful procedures due to the *revelation principle*.

In studying truthful and fair resource allocation, we adopt a computational lens. When implementing resource allocation algorithms in practice, there can be two computational barriers. First, if agent preferences are very rich, then it may not be possible for agents to completely communicate their preferences. Second, it is necessary to ensure that our resource allocation algorithms are computationally tractable, i.e., run in time polynomial in the size of a natural input representation. Our discussion of fair resource allocation focuses on restricted families of valuations (mappings from allocations to values) and is motivated by the first computational barrier. In our discussion of truthful mechanisms computation is relevant in two ways: we use computational approaches to design mechanisms, and this in turn helps to address settings where polynomial time computation is a barrier to some otherwise truthful mechanisms.

1.1 Fairness and Cake Cutting

Our discussion of fair resource allocation algorithms focuses on fair division or the cake cutting problem. In this setting, there is a single divisible good to be allocated among a set of agents. The good is heterogeneous, in the sense that agents have different values for the different parts of the good. As a concrete example, the divisible good could be time on a compute server or a newly cleared plot of land. The typical goal in cake cutting is to give a procedure that finds a fair allocation of the good. There are various notions of fairness, but informally, *proportionality* requires that each agent receive at least $1/n$ of its value for the entire good if there are n agents, *envy-freeness* requires that each agent prefer its own allocation to that of any other agent, and *equitability* requires that all agents receive the same value from their allocations.

Work on cake cutting dates back to Steinhaus in the 1940s [Steinhaus, 1948], and has traditionally been studied by mathematicians, economists, and political scientists. More recently, cake cutting has gained the attention of computer scientists and the artificial intelligence (AI) community (Procaccia [2009], Chen et al. [2010], Zivan et al. [2010], Caragiannis et al. [2011], Cohler et al. [2011], Brams et al. [2012a], Maya and Nisan [2012], Brânzei and Miltersen [2013], Brânzei et al. [2013], Kurokawa et al. [2013]). The survey by Procaccia [2013] provides a nice summary of recent work. Cake cutting is of interest to AI due to its potential importance in multi-agent resource allocation [Chevaleyre et al.,

2006]. Additionally, computer science provides a new and interesting perspective on cake cutting by thinking about the representation of agent preferences and the computational complexity of cake cutting.

Slightly more formally, there is a set of n agents and the cake is represented by the interval $[0, 1]$. Each agent has a value density function v_i over the interval. The agent's value for some set of disjoint subintervals is the integral of its value density function over those subintervals. This definition makes agent's valuation functions additive and non-atomic so that agents derive no value from receiving a subinterval $[x, x]$ consisting of a single point. Agent valuations are assumed to be normalized so that each agent receives value 1 for the entire cake. This assumption is typically without loss of generality and makes our exposition simpler, and we will discuss whether our results extend to the more general unnormalized case. A cake cutting procedure or algorithm outputs an *allocation* $A_1, \ldots, A_n$ where A_i represents a set of disjoint subintervals of $[0, 1]$ and $A_i \cap A_j = \emptyset$. An allocation is *proportional* if each agent receives value at least $1/n$, *envy-free* (EF) if each agent values its assigned piece weakly more than the pieces of other agents, and *equitable* (EQ) if all agents have the same value for the pieces they are assigned, i.e., for any two agents i, j, agent i's value for A_i is the same as agent j's value for A_j.

1.1.1 Cake Cutting under Restricted Valuations

The classic cake cutting literature operates under the assumption that agents can have any integrable value density function. As a result, classic cake cutting procedures never fully pin down agents' valuations but instead guarantee that the computed allocation will have the desired fairness properties with respect to any value density function consistent with the agents' actions and responses to queries. As a simple example of a classic cake cutting procedure, consider the Cut and Choose procedure for two agents. Agent 1 splits the cake into two pieces of equal value, and agent 2 chooses the more preferred piece, leaving the other piece to agent 1. While the final allocation is fair, we do not know agent 2's exact values for these two pieces, and we don't have much information about the agents' exact value density functions on subintervals of each of these two pieces.

A key conceptual contribution of this dissertation, motivated by thinking about the representation of agent valuations as inputs to a computational procedure, is the introduction of the study of cake cutting when agents have restricted preferences. Indeed, the study of restricted valuations can help with gaining a better understanding of the difficulties of cake cutting as well as make it easier for cake cutting algorithms to gain traction in practice.

Specifically, we consider cases where agent's value density functions have very specific forms. We consider the *piecewise uniform* case where the density function is either 0 or some constant c, the *piecewise constant* case where the value density function is piecewise constant, and the *piecewise linear* case where the value density function is piecewise linear.

Though these families of valuation functions are restrictive, there are natural settings where they might be appropriate. Piecewise uniform valuations capture the setting where an agent has intervals that are preferred and intervals that yield no value, but the agent has the same marginal value for the preferred intervals. If the divisible good is being allocated is time on a shared compute server, then an agent could plausibly have such preferences if there are some scheduling constraints that prevent it from making use of certain time intervals but the agent is indifferent among intervals without scheduling conflicts. Similarly, if the divisible good being allocated is land, then agents may have simple preferences that makes plots of land acceptable only if they have access to a canal. Piecewise uniform valuations capture these simple settings. In comparison, though piecewise constant and piecewise linear valuations are also restrictive, they can be used to approximate general valuations and so in a sense are not restrictive at all.

It is tempting to draw an analogy between agents with piecewise constant valuations and an indivisible goods setting where there is an indivisible good for each subinterval on which agents' value density functions are constant. Then allocations of cake can be associated with probabilities of receiving each of these indivisible goods. The problem with this analogy is that piecewise constant valuations allow agents to control the identity and number of indivisible goods available whereas in indivisible good settings it is typically assumed that the goods are pre-defined and cannot be affected by agents' preferences. We discuss this point further in Chapter 6 when we compare our setting with the random assignment problem.

We have thus far discussed the disadvantage of these restricted families, namely that they are not as expressive as general valuations. However, these restricted families are very useful for the following reason:

It is possible to succinctly communicate these preferences, thereby allowing for a new class of cake cutting algorithms which operate directly on the underlying value density functions.

For example, a piecewise uniform valuation can be communicated by listing the endpoints of an agent's intervals of interest. Similarly, a piecewise constant valuation can be communicated by listing the endpoints of intervals on which an agent's density function is

constant and providing the value of the density function on each of these intervals. Likewise, a piecewise linear valuation can be communicated by listing the endpoints of intervals on which an agent's density function is linear and providing the slope and intercept of the density function on each of these intervals.

In the context of mechanism design (to be discussed in the next section), assuming restricted families of valuation functions allows for the study of *direct* mechanisms. Indeed, the inspiration for studying restricted families of valuations came from our study of truthful cake cutting (Chapter 6), but it turns out that allowing the algorithm to operate on the exact valuations also gives rise to many other interesting questions.

1.1.2 Welfare Maximization in Cake Cutting

With restricted families of valuations and assuming that the algorithm has access to agent's exact valuations, a natural question to ask is:

How should we choose among the set of fair allocations for various notions of fairness?

This question is less natural under the classic cake cutting model since an allocation may be better or worse depending on the particular value density function that is consistent with the agents' responses to queries made by the algorithm. A natural tie-breaker is to consider the *social welfare* or total value created by an allocation (we will often drop the modifier social and just refer to social welfare as welfare). An allocation is *maxsum fair* if it has the highest total welfare among allocations that are fair. For instance, a maxsum EF allocation is an allocation that has the highest total welfare among all EF allocations.

Since welfare involves summing values across agents, the assumption that agent value density functions are normalized so that $V_i([0,1]) = 1$ is no longer without loss since the maxsum fair allocations will differ before and after normalization. In our study of maxsum cake divisions we assume normalization, but the positive algorithmic results extend naturally to settings where valuations are not normalized and the negative impossibility results are only strengthened by expanding the set of possible valuations.

Our first contribution is to consider algorithms for computing maxsum EF allocations. When valuations are piecewise constant, we provide a linear program (LP) that finds a maxsum EF allocation in time polynomial in the number of bits needed to specify the agents' value density functions. For the case of two agents and piecewise linear valuations, we show that exactly computing a maxsum EF allocation is not possible for polynomial time algorithms. The reason is not due to computational complexity, but due to the fact that there are cases where every maxsum EF allocation requires cuts at irrational points in $[0,1]$

even when agents' value density functions can be expressed using rational numbers. We then give an algorithm that computes an approximately EF allocation with welfare at least as great as any maxsum EF allocation for two agents. This algorithm runs in time polynomial in the input size and in $O(\log(1/\epsilon))$ where ϵ controls the level of approximate envy-freeness. We then consider general value density functions under a Lipschitz continuity condition and leverage our piecewise constant result to give an algorithm that is approximately EF and has welfare that approximates any maxsum EF allocation. This algorithm runs in time polynomial in $1/\epsilon$ where ϵ controls the amount of deviation from exact envy-freeness and the deviation from the welfare of any maxsum EF allocation. Though we focus our exposition on maxsum EF allocations, our techniques extend to maxsum proportional and maxsum EQ allocations (allocations that have maximal welfare among all proportional and all EQ allocations respectively).

Having considered the problem of computing maxsum fair allocations, we next turn our attention to understanding properties of these allocations. The goal is to shed light on the relative qualities of these different maxsum fair allocations and help to choose among them for particular applications. Specifically, we consider whether these maxsum fair allocations are *Pareto-efficient*, meaning that there are no allocations that make all agents weakly better off and at least one agent strictly better off. When considering Pareto-efficiency we allow the allocations we compare against to be unfair, as maxsum fair allocations are trivially Pareto-efficient if we limit our comparisons to fair allocations. We find, surprisingly, that there exist piecewise constant valuations for three agents in which every maxsum EF allocation is not Pareto-efficient. For maxsum EQ or maxsum EF+EQ allocations the situation is even more dire, as there piecewise uniform valuations where all maxsum EQ and maxsum EF+EQ allocations are not Pareto-efficient. In contrast, it is an easy exercise to argue that all maxsum proportional allocations are Pareto-efficient (for any valuations).

The second result along these lines compares the social welfare of maxsum EQ and maxsum EF allocations. We prove that for piecewise linear valuations, the welfare of maxsum EF allocations is always weakly greater than the welfare of maxsum EQ allocations. We also use this result to show an approximate version of this result for general valuations. This suggests that an exact result holds for the general case as well, but the proof of this remains an open question.

1.1.3 Expressiveness

We next turn our attention to a different aspect of cake cutting while still operating in a direct revelation model on restricted families of valuations. One assumption that is made in cake cutting is that agent valuation functions are additive. This is a consequence of the definition of agent values as the integral of a value density function. One implication of this is that agents gain positive value from disjoint intervals that are each very small in size. While this idealized model may hold in certain settings, there are some settings where it is unrealistic. For instance, when allocating time on a compute server, there are significant context switching costs that make many disjoint chunks of compute time less valuable. Similarly, in the allocation of television advertising slots, slots that are less than 10 seconds in length may not be useful. As a result, we undertake the study of cake cutting in a setting where agents have piecewise uniform valuations with a minimum length parameter λ. This parameter states that agents have no value for intervals less than λ in length and prevents the algorithm from allocating many disjoint intervals that are each very tiny.

While proportional and EF allocations always exist in the traditional cake cutting model, the same is no longer true when agents have a minimum length parameter. As a simple example, consider the case where both agents have a minimum length parameter of 1. In this case, an agent must receive the entire interval to have positive value, leaving nothing for the other agent. Hence, exact proportionality is impossible in this setting. However, we provide a polynomial time algorithm that guarantees an approximately proportional allocation where the approximation is additive. Our algorithm is a carefully constructed generalization of a proportional cake cutting procedure for n agents, and we show that it attains the optimal additive approximation guarantee. The approximation is worse for larger values of λ, capturing the fact that it is more difficult to satisfy agents with large values of λ. We prove that the additive approximation we provide is essentially optimal among all algorithms that give an additive worst-case approximation guarantee.

We then consider proportionality and envy-freeness together in this model with a minimum length parameter. For two agents, we give a polynomial time algorithm that finds an allocation that satisfies the best possible approximation to proportionality while preserving exact envy-freeness. The algorithm is quite intricate and specific to the two agent case and finding a generalization for any number of agents is likely quite challenging.

1.2 Truthfulness and Mechanism Design

Having touched on our contributions to fair resource allocation, we discuss truthfulness and mechanism design (we will also consider truthfulness and fairness together for cake cutting). We provide a more rigorous treatment of mechanism design in Chapter 2, but we give a brief introduction here. In order to study truthfulness of resource allocation algorithms, we must clearly define what is meant by truthfulness. Our requirement of truthfulness arises from the economic theory of *mechanism design*. In mechanism design, there are have n agents, each agent has a type, there is a set of possible decisions. An agent's type specifies how much utility it derives from every possible decision. In full generality, a mechanism allows the agents to send a message to the mechanism, and the mechanism selects a decision based on the messages passed by the agents. In this dissertation, we focus on *direct mechanisms*, or mechanisms where the messages are direct reports of an agent's type. A mechanism is therefore a mapping from a set of reported types, one for each agent, to a decision.

It is often the case that the decision can be broken down into an outcome component and a payments component. The payments component may or may not be a part of the decision. For example, in the cake cutting setting, the set of possible outcomes consists of the permitted allocations of the cake and there are no payments, so the set of outcomes coincides with the set of possible decisions. On the other hand, consider a single-item auction. The possible outcomes consist of which agent the item is given to, and this is a setting with payments, so the decision made by the mechanism includes the outcome and a payment for each agent. When there are payments, we assume that agents have quasi-linear utilities, so that an agent's utility for receiving some outcome and paying some amount is equal to its value for the outcome minus the required payment. We often use the term algorithm interchangeably with outcome rule, since the outcome rule is typically solving some sort of optimization problem to maximize an objective given the agents' reported valuations.

The goal of mechanism design is to make a good decision, where the quality of the decision is evaluated with respect to the agents' true types. It is assumed that agents will behave in a way that maximizes their utility and will not not necessarily report their true types. A *strategy* maps an agent's true type to a reported type. Given strategies of other agents, an agent has a set of strategies that are best responses. An *equilibrium* is a set of strategies where each agent's strategy is a best response to the strategies of other agents. To understand the quality of the decisions of a mechanism, it is necessary to identify the equilibria of the mechanism.

In this dissertation, we focus on *truthful* mechanisms, i.e., mechanisms with equilibria where each agent's strategy is to report its true type. In truthful mechanisms, it is easy to analyze the quality of the mechanism, as we can just measure the quality of the decision with respect to reported types (which we can assume are the same as the true types) and do not need to reason about the agent's true underlying types. It is also simple for agents to participate in truthful mechanisms since agents need only report their true types and need not strategize about their reports. Furthermore, it turns out that there is a sense in which studying truthful mechanisms is without loss due to what is known as the *revelation principle*.

There are various notions of truthful mechanisms depending on definition of what constitutes an equilibrium. The strongest notion of equilibrium we consider, *dominant strategy equilibrium*, requires that agents' best responses do not depend on the strategies of other agents. Mechanisms with a truthful dominant strategy equilibrium are known as *dominant strategy incentive compatible* (DSIC). In these mechanisms, reporting truthfully maximizes an agents' utility regardless of the actual types of the other agents. A weaker notion of equilibrium is *Bayes-Nash equilibrium*, which applies to settings where agents' types are drawn from a commonly known distribution. In Bayes-Nash equilibrium, agent strategies are best responses in expectation to the strategies of other agents, where the expectation is taken over the possible types of the other agents. Mechanisms with a truthful Bayes-Nash equilibrium are known as *Bayes-Nash incentive compatible* (BIC). In these mechanisms, reporting truthfully maximizes an agents' expected utility, assuming that other agents' types are randomly drawn from the common knowledge distribution and that other agents report truthfully. This is a weaker notion than DSIC because reporting truthfully is only guaranteed to be a best response in expectation and only if other agents' report truthfully.

1.2.1 Truthful Cake Cutting

Our first contribution to truthful resource allocation revisits the cake cutting problem with restricted valuation functions. In particular, we examine the case of piecewise uniform valuations and seek a mechanism that is both truthful and fair in this setting. Recall that a mechanism in this setting is simply an algorithm that computes an allocation given the agents' reported types.

Most prior work on cake cutting ignores strategic issues and simply assumes that agent valuations are either publicly known or truthfully revealed. Indeed, our other contributions to the cake cutting literature have this flavor. Work that did consider strategic issues

examines a very weak notion of truthfulness. The notion considered in previous papers by Brams et al. [2006] and Brams et al. [2008] assumes that an agent will report its true valuation rather than lie if there *exist* valuations of the other agents such that reporting truthfully yields at least as much value as lying. In the words of Brams et al.,

> The players are risk-averse and never strategically announce false measures if it does not guarantee them more-valued pieces. ... Hence, a procedure is strategy-proof if no player has a strategy that dominates his true value function [Brams et al., 2008, page 362]

We depart from this prior work and instead attempt to find mechanisms that are DSIC. In contrast to the notion of of truthfulness studied by Brams et al., DSIC requires that truthfully revealing one's type weakly dominates any other type report for all possible reports of other agents.

We also depart from prior work by assuming that agents have piecewise uniform valuations. This allows us to consider direct mechanisms, whereas prior work on strategic issues operates in the classic cake cutting model where agent valuations are never fully revealed. Indeed, our main result depends crucially on the piecewise uniform assumption and knowing the agents' exact valuations.

Our main result is a deterministic mechanism that is DSIC, proportional, and EF. In addition, this mechanism computes a Pareto-efficient allocation (relative to the reported types) and runs in polynomial time. We also discuss randomized mechanisms and provide mechanisms that are *truthful in expectation* and universally fair. Truthfulness in expectation requires that an agent maximizes its expected utility by reporting truthfully where the randomness is over the random choices of the mechanism. Universal fairness requires that the computed allocation is always fair, regardless of the random choices of the mechanism. We give randomized mechanisms that are truthful in expectation and universally fair for piecewise linear valuations. The mechanisms rely on the existence of *perfect partitions* which divide the cake into n pieces such that every agent has value $1/n$ for every piece. Though these partitions seem very special, their existence has been known since the 1940s [Neyman, 1946]. These proofs are non-constructive, though, and our contribution is to provide explicit constructions for agents with piecewise linear valuations.

1.2.2 Combinatorial Auctions

Combinatorial auctions (CAs) are a focus of the rest of our discussion of truthful resource allocation, though we point out where our results are also applicable to more general settings. In a combinatorial auction, we wish to allocate a set of items to a set of agents, and

agents hold private valuations over subsets or bundles of items. Combinatorial auctions are useful (when compared with running separate single item auctions) when agent valuations exhibit complementarities, i.e. the value for a bundle of items exceeds the sum of the values of the individual items in the bundle.

The canonical example of complementary items is a left shoe and a right shoe, but many real world resource allocation settings exhibit complementarities as well. For example, in national wireless spectrum auctions in the United States, the individual items are bands of spectrum in a particular region [Cramton, 2002]. A national cellular provider like Verizon would exhibit strong complementarities for having spectrum across all regions (since it seeks to be a national provider). Indeed, combinatorial auctions have found application in the allocation of congested landing slots at airports [Ball et al., 2006], the procurement of service providers for operating bus routes in London [Cantillon and Pesendorfer, 2006], and various industrial procurement settings [Caplice and Sheffi, 2006, Bichler et al., 2006, Sandholm, 2013].

Combinatorial auctions determine an assignment of items to each agent and charge each agent a payment. We assume that we are in a quasi-linear setting where agent utilities are equal to their value for the items they receive minus the payment they make.

Combinatorial auctions provide a canonical example where the economic theory of mechanism design is at odds with limited computational power and serve as a motivation for the study of *computational mechanism design*. Computational mechanism design adds the constraint that a mechanism needs to be computationally tractable, both in terms of eliciting the valuations from the agents and in terms of computing the outcome and payments.

Slightly more formally, there is a set of n agents and a set of m items. Each agent has a private type which determines its value for different possible subsets of items. An agent's type provides a value for every possible bundle of items. An *allocation* assigns a subset of items to each agent, ensuring that each item is allocated at most once. The direct mechanism design problem is to find an outcome rule that maps reported types to an allocation and a payment rule that maps reported types to a payment for each agent.

Computational Challenges

Combinatorial auctions present two computational challenges. The first is that in full generality, reporting an agent's valuation may require the specification of 2^m real numbers, each representing a value for a possible subset of items the agent receives. This *preference elicitation* problem makes direct mechanisms infeasible if agents do indeed have such complex

preferences. One approach to circumvent these issues is to consider indirect mechanisms (see Parkes [2001]). A second approach, which is the one we use in this dissertation, is to focus on restricted classes of preferences. Specifically, we examine cases where agents are single-minded or multi-minded. What this means is that agents have a set of target bundles in mind. If the agents receive a subset of items that does not include any target bundle, then the agent receives value 0 for that subset. Otherwise, the agent receives a value equal to the largest included target bundle. As a result, agents need only communicate their target bundles and their values for those bundles, and this pins down their entire valuation function.

The second computational challenge is that optimizing over the set of possible allocations can be a computationally difficult problem. As an example, consider the case where the outcome rule maximizes social welfare. Finding a welfare-maximizing allocation is NP-hard in most combinatorial auction settings, including the single-minded setting discussed above [Rothkopf et al., 1998, Lehmann et al., 2002]. If computational constraints were not important, then the class of Vickrey-Clarke-Groves (VCG) mechanisms are incentive compatible and have an outcome rule that exactly maximizes social welfare [Vickrey, 1961, Clarke, 1971, Groves, 1973]. The key to incentive compatibility in VCG mechanisms is the design of a payment rule that aligns agents' incentives with the social welfare of the computed allocation. Unfortunately, the VCG mechanism cannot be used if polynomial time computation is required due to the NP-hardness of finding an allocation that exactly maximizes social welfare. A natural approach is to replace the outcome rule with an algorithm that approximates the optimal social welfare while using a payment rule that is similar in spirit to the VCG payment rule. The Clarke-Pivot version of the VCG mechanism can be thought of as charging agent i the externality it imposes on other agents. The same idea can be adapted to any outcome rule. Unfortunately, this new mechanism is no longer incentive compatible. In fact, the outcome rule that approximately maximizes welfare may not satisfy the necessary *monotonicity* properties for there to exist payment rules that form an incentive compatible mechanism when combined with the outcome rule.

One approach to this problem is to develop polynomial time algorithms that find allocations whose welfare is provably within some factor of the optimal welfare while simultaneously satisfying the required monotonicty properties for there to exist incentive compatible payment rules. Specializing to the setting of single-minded combinatorial auctions, Lehmann et al. [2002] take this approach and give a simple greedy algorithm that satisfies the required monotonicity properties and guarantees that the computed allocation

has welfare at least $O(1/\sqrt{m})$ times the optimal welfare. Similarly, Mu'alem and Nisan [2008] devise algorithms that satisfy the monotonicity properties and compute an allocation with welfare at least $O(1/(\epsilon\sqrt{m}))$ times the optimal for any fixed $\epsilon > 0$ with runtime that is exponential in $1/\epsilon^2$. These approaches are analytical in the sense that these approximation algorithms are completely specified, and the proofs of monotonicity and worst-case approximation involve reasoning about the specifics of the approximation algorithm.

In this dissertation, we focus on more computational approaches, where we do not attempt to give a specification of the approximation algorithm, but rather we give computational procedures that produce algorithms with certain desirable properties. Specifically, we provide an approach that computationally modifies Branch-and-Bound (BnB) search for solving integer programs and makes it usable as the outcome rule for a truthful mechanism. We also develop an approach that given an outcome rule, uses machine learning to find a payment rule that is approximately incentive compatible when paired with the outcome rule. We discuss existing work related to these directions in the next section.

1.2.3 Computational Approaches to Mechanism Design

Conitzer and Sandholm [2002] introduced the agenda of *automated mechanism design* (AMD) and formulated mechanism design as the search for an allocation rule and (possibly) a payment rule among a class of rules satisfying incentive constraints. While the basic idea was of course already familiar from the seminal work of Myerson [1981], a novel aspect of the work of Conitzer and Sandholm is that it explicitly represents a mechanism as a mapping from type profiles, where type spaces are assumed to be discrete, to outcomes and payments. This explicit representation makes AMD intractable in general because the number of type profiles is exponential in the number of agents, and possibly also in other natural parameters of a problem such as the number of items in a combinatorial auction. One approach that was adopted to make AMD more tractable is to search through a parameterized space of incentive compatible mechanisms [Likhodedov and Sandholm, 2005, Guo and Conitzer, 2010b]. More recently, advances in AMD have been made in application to the design of revenue optimal mechanisms and by considering BIC rather than DSIC [Cai et al., 2012a].

A different computational approach assumes that access to some approximation algorithm that we wish to use as our outcome rule. The challenge, however, is that the approximation algorithm may not have the proper monotonicity properties to guarantee existence of incentive compatible payment rules.

For the target of DSIC mechanisms, Briest et al. [2005] give a construction that converts

pseudopolynomial time algorithms that maximize welfare welfare into incentive compatible fully polynomial time approximation schemes (FPTAS).[1] Their insights also yield new truthful mechanisms for settings without pseudopolynomial time algorithms, though these mechanisms do not have the flavor of converting a non-truthful algorithm; rather, they are fully specified algorithms like those of Lehmann et al. [2002] and Mu'alem and Nisan [2008].

For the target of randomized truthful in expectation mechanisms, Lavi and Swamy [2011] give a construction that converts a non-truthful approximation algorithm into a randomized truthful in expectation mechanism that preserves the worst-case approximation guarantee (in expectation) of the original rule. They require that the optimization problem can be written as an integer program and that there exists an α-approximation algorithm that also bounds the integrality gap of the LP relaxation of the problem by α. Their construction yields a truthful in expectation mechanism with approximation guarantee of $O(1/\sqrt{m})$ when applied to (general) CAs. Dughmi and Roughgarden [2010] give a construction that yields a randomized truthful in expectation mechanis (with outcome rules which are an FPTAS) for any welfare maximization problem that has an FPTAS and can be encoded as a set packing problem.

For the target of BIC mechanisms, Hartline and Lucier [2010] and Hartline et al. [2011] provide a general approach, for both single-parameter and multi-parameter domains, for converting any algorithm into a BIC mechanism with essentially the same welfare as the original algorithm. Related to this, Bei and Huang [2011] also tackle multi-parameter discrete domains and give a construction that converts any algorithm into an ϵ-BIC while losing a small amount in welfare. Recent work by Cai et al. [2012b] give a construction that takes an algorithm that maximizes welfare and converts it into a revenue optimal BIC mechanisms. This construction is computationally tractable when agents are additive (their value for a bundle of items is the sum of their value for each item) and independent (an agent's type is not correlated with other agent types). Cai et al. [2013] extend this construction to convert algorithms that only approximately maximize welfare into BIC mechanisms that approximate the revenue optimal BIC mechanism with the same approximation factor.

Conitzer and Sandholm [2007] propose an *incremental mechanism design* approach where a mechanism is incrementally made more truthful by finding and correcting violations of truthfulness. In a similar vein, Parkes [2009] proposes an agenda of *heuristic*

[1] A pseudopolynomial time algorithm runs in time that is polynomial in the numeric value of the input rather than the number of bits needed to represent the input. An FPTAS is a scheme that given a desired error ϵ gives a solution within a factor $1-\epsilon$ of optimal and runs in time polynomial in $1/\epsilon$ and the size of the input to the algorithm.

mechanism design where heuristic algorithms are modified for the purposes of mechanism design. The modifications may not be black box (like those in the previous paragraph) in the sense that they can be specific to the particular algorithm. These algorithms may not have worst case guarantees, but they work well in practice. Along these lines, Parkes and Duong [2007] and Constantin and Parkes [2009] apply a so-called "computational ironing" approach to online stochastic combinatorial optimization (OSCO). Our work on monotone branch and bound search is closely related to this agenda (see next section and Chapter 8). Also thematically related to heuristic mechanism design is the GrowRange method of Parkes and Schoenebeck [2004], which provides an anytime algorithm for welfare optimization in general CAs by expanding the range of a VCG-based algorithm, while allowing for a time-based interruption by the center (although without providing full incentive compatibility).

1.2.4 Monotone Branch and Bound Search

We make two contributions to work on computational approaches to mechanism design. The first investigates the setting of known single-minded CAs. A known single-minded CA is the same as a single-minded CA, with the added assumption that agents' single target bundles are publicly known. The only private information is therefore an agent's value for its target bundle, thereby transforming the problem into a *single-dimensional* mechanism design problem. While the known single-minded assumption is restrictive, Lehmann et al. [2002] describe a pollution rights auction where companies bid for the right to emit certain chemicals into the air, and the pollution profiles of the companies are known. They also describe communication network settings where bidders own nodes in the network and wish to connect their nodes. If there is only a single path available between any pair of nodes, then bidders are single-minded. If it is also public knowledge which companies own which pairs of nodes, then this becomes a known single-minded setting.

Even in known single-minded CAs, computing the exact welfare-maximizing allocation is NP-hard, thereby preventing the application of the VCG mechanism computational tractability is required. As discussed in the previous section, Lehmann et al. [2002] and Mu'alem and Nisan [2008] tackle this problem and give incentive compatible mechanisms that are computationally tractable, have worst-case guarantees on welfare, and satisfy the requirements of incentive compatibility. These mechanisms rely on relatively simple allocation algorithms, such as a greedy algorithm which ranks bundles by score and greedily assigns bundles in order in decreasing score, or an allocation function which iterates over all

feasible allocations that allocate fewer than some number of agents. However, if incentives were not a concern, there are more sophisticated algorithms such as Branch-and-Bound (BnB) search, which can efficiently find optimal solutions to the winner determination problem on typical instances.

Specifically, we consider a variant of BnB search where the search continues until a solution is found that is within a factor $\gamma < 1$ of optimal. This is possible due to the various bounds that are maintained during the BnB search procedure. While running BnB search to completion is monotone in each agent's reported value, running to a tolerance $\gamma < 1$ is no longer monotone. As a result, we devise a procedure that performs sensitivity checking on the BnB search tree to check for monotonicity violations. If violations are found, i.e. an agent is allocated for some initial report but deallocated for a higher report (recall that we are in a single minded setting where an agent's report is just a single number so a higher report is well-defined), we decide not to allocate the agent at its initial report. The core technical contribution is the design of an optimized sensitivity checking procedure that takes advantage of the structure in a BnB search tree.

We perform experiments on instances from the Combinatorial Auctions Test Suite [Leyton-Brown et al., 2000], and we find that the best parameterizations of our procedure can outperform the existing methods of Lehmann et al. [2002] and Mu'alem and Nisan [2008] in terms of welfare while taking less runtime than running BnB search to completion. Additionally, our monotone BnB is fully parallelizable (the sensitivity checking can consider each agent in isolation), thereby further reducing the runtime cost.

1.2.5 Learning Payment Rules

Our second contribution to computational approaches to mechanism design considers the setting where there is some outcome rule that we wish to use but we do not have a payment to pair with the outcome rule. For instance, we might have some heuristic algorithm that finds allocations with good welfare, or we might have an algorithm that finds allocations that maximize a non-standard objective function such as egalitarian welfare (maximize the minimum value). Instead of imposing incentive compatibility as a hard constraint (which might require us to modify the outcome rule), we relax the requirement of exact incentive compatibility. Taking the outcome rule as given, we attempt to find a payment rule with good incentive properties.

Specifically, we adopt statistical machine learning to infer payment rules that minimize agents' *expected ex post regret*. The ex post regret (or just *regret* where it causes no confu-

sion) of an agent for truthful reporting in a given instance is the maximum amount by which its utility could increase through a misreport holding constant the reports of others. The expected ex post regret is the average ex post regret over all agents and all preference types, calculated with respect to a distribution on types. In addition to minimizing expected ex post regret, we enforce an *agent independence* property on the payment rule that ensures that (small) changes in an agent's report that do not change the computed outcome will not be beneficial.

At the core of our approach lies a relationship between incentive compatible mechanisms and multi-class classifiers. Given an outcome rule, the related multi-class classification problem is to predict, given a type profile, the outcome that the outcome rule produces when given the type profile. If we have an *admissible* classifier, then based on the parameters of the classifier, it is possible to derive a payment rule to pair with the provided outcome rule. We formally show that an exact, admissible classifier yields a payment rule that is incentive compatible when paired with the provided outcome rule, and that an incentive compatible mechanism ensures the existence of an exact, admissible classifier. We also show that an admissible classifier that minimizes a particular generalization error yields a payment rule that minimizes expected regret.

We train our admissible, discriminant-based classifiers by adapting structural support vector machines [Joachims et al., 2009] to our mechanism design setting. We implement our techniques and apply them to three problem domains where incentive compatible mechanisms are not known. The first setting is multi-minded CAs. We seek to maximize welfare, but we adopt a greedy heuristic algorithm as the outcome rule since exactly maximizing welfare is NP-hard. The second setting is CAs where agents have positive, k-wise dependent valuations [Conitzer et al., 2005, Chevaleyre et al., 2008]. Exact welfare-maximization is NP-hard in this setting as well, so we also adopt a greedy outcome rule that attempts to maximize welfare, and we seek a payment rule that minimizes regret. The final setting is an assignment setting where there are as many items as agents and the goal is to find an assignment of exactly one item to each agent. Here we adopt an egalitarian outcome rule that computes an assignment that maximizes the minimum value attained by any agent. Because of the egalitarian objective, incentive compatible mechanisms are not known for this setting. Our results show that we are able to find payment rules with better regret than natural VCG-based payment rules. These VCG-based payment rules adopt a similar approach of charging an agent the externality it imposes on other agents, except that the externality is measured with respect to an outcome rule that does not necessarily exactly

maximize social welfare.

Several interesting complications arise in learning payment rules, and we provide solutions to each of these. For CAs, the space of possible outcomes is exponential in the number of items. This creates an exponentially large number of constraints in the structural support vector machine formulation of the training problem. Solving the training problem requires the existence of a polynomial time separation oracle. We do note that training can be performed offline, so the runtime requirements for training may not be as stringent as those for computing the actual payments given agent reports. For multi-minded CAs, we do not have such a polynomial time separation oracle and are thus limited to training payment rules for small problem sizes. On the other hand, by adopting positive k-wise dependent valuations, we are able to formulate a polynomially sized training problem by drawing connections to the literature on tractable maximum a-posteriori (MAP) estimation for associative Markov networks [Taskar et al., 2004]. Another problem that arises is that the learned payment rules may violate *individual rationality* (IR) in that an agent may be charged more than its value for the outcome. We present several ways of mitigating IR violation by modifying the training problem to learn payment rules with less IR violation and providing several IR fixes that modify the learned payment rule. We experimentally show that these solutions are effective in the domains we study.

1.3 Outline

This dissertation can be thought of as consisting of two parts. In the first part, we consider cake cutting and fairness, without considering truthfulness. In the second part, we focus on mechanism design problems and truthfulness. In this second part, we first discuss truthfulness in a cake cutting setting and then proceed to develop two computational approaches to mechanism design.

Chapter 2 provides background on resource allocation and mechanism design. We cover some main results from mechanism design theory and provide precise definitions of truthfulness and motivations for the study of truthful mechanisms.

Chapter 3 formally defines the cake cutting problem and reviews some classic results from the cake cutting literature. We discuss a direct revelation model which departs from the classic model of cake cutting, and introduce the study of cake cutting under restricted families of valuation functions. Chapters 4 and 5 consider questions related to the direct revelation model and restricted families of valuations. In particular, Chapter 4 examines

welfare-maximizing fair allocations, while Chapter 5 investigates a model that extends the expressiveness of piecewise uniform valuations.

Chapter 6 begins our investigation of truthfulness and mechanism design. We consider the cake cutting problem when agents have piecewise uniform valuations. Chapter 7 introduces some necessary background on combinatorial auctions (CAs). We introduce the problem, touch on real world applications, and explain how combinatorial auctions demonstrate the tension between a purely economic approach to mechanism design and computational tractability. Chapters 8 and 9 describe our two computational approaches to mechanism design, with Chapter 8 describing our monotone BnB search procedure and Chapter 9 describing our procedure for learning payment rules using machine learning.

Chapter 10 concludes and discusses some possible future directions.

1.4 Bibliographic Notes

The results in this dissertation are largely based on the following previously published papers.

1. Yuga J. Cohler, John K. Lai, David C. Parkes, and Ariel D. Procaccia. Optimal envy-free cake cutting. In *AAAI*, 2011. (Section 4.2)

2. Steven J. Brams, Michal Feldman, John K. Lai, Jamie Morgenstern, and Ariel D. Procaccia. On maxsum fair cake divisions. In *Proceedings of the 26th AAAI Conference*, 2012a. (Section 4.3)

3. Ioannis Caragiannis, John K. Lai, and Ariel D. Procaccia. Towards more expressive cake cutting. In *Proceedings of the 22nd IJCAI*, 2011. (Chapter 5)

4. Yiling Chen, John K. Lai, David C. Parkes, and Ariel D. Procaccia. Truth, justice, and cake cutting. In *Proceedings of the 24th AAAI Conference*, pages 756–761, 2010. and
   Yiling Chen, John K. Lai, David C. Parkes, and Ariel D. Procaccia. Truth, justice, and cake cutting. *Games and Economic Behavior*, 77(1):284–297, 2013. (Chapter 6)

5. John K. Lai and David C. Parkes. Monotone branch-and-bound search for restricted combinatorial auctions. In *Proceedings of the 12th EC*, pages 705–722, 2012. (Chapter 8)

6. Paul Dütting, Felix A. Fischer, Pichayut Jirapinyo, John K. Lai, Benjamin Lubin, and David C. Parkes. Payment rules through discriminant-based classifiers. In *Proceedings of the 12th EC*, pages 477–494, 2012. (Chapter 9)

Chapter 2

Resource Allocation and Mechanism Design

This chapter formally introduces the model for resource allocation and provides necessary background on mechanism design.

2.1 Formal Model

A resource allocation problem is given by a set $N = \{1, 2, \ldots, n\}$ of *agents* that interact to make a collective decision. $\mathcal{D}$ denotes the set of possible decisions and d denotes a member of this set. Each agent $i \in N$ is associated with a *type* θ_i from a set Θ_i of possible types. There exists a *utility function* $u_i : \Theta_i \times \mathcal{D} \to \mathbb{R}$ for each agent i that maps a possible type and decision to the agent's utility for that decision.

We write $\theta = (\theta_1, \ldots, \theta_n)$ for a profile of types for the different agents, $\Theta = \bigtimes_{i \in N} \Theta_i$ for the set of possible type profiles, and $\theta_{-i} \in \Theta_{-i}$ for a profile of types for all agents but i. We are interested in procedures or algorithms that take the agent's types and compute decisions.

Definition 2.1.1. A *social choice function* $f : \Theta \to \mathcal{D}$ maps a profile of types θ to a decision d.

2.1.1 Concrete Examples

To make the setting concrete, consider the two main topics we discuss in this these: cake cutting and combinatorial auctions. Chapters 3 and 7 provide a more formal and detailed discussion, but here we give a brief overview of how these problems map to the formal model

we have just defined. In cake cutting, there is a single divisible good being allocated. The possible decisions $\mathcal{D}$ are assignments of parts of the divisible good to each agent, making sure that the parts received by each agent are disjoint. The agents' types specify how much utility they derive from different parts of the good. For instance, an agent of type θ_i^1 might prefer only the vanilla part of the cake; on the other hand, an agent of type θ_i^2 might prefer only the cherry. The combinatorial auction (CA) problem involves allocating a set of items to the set of agents. The possible decisions $\mathcal{D}$ consist of an assignment of sets of items to each agent as well as a payment that each agent has to make. The agents' types specify how much utility they derive from a particular assignment of items and a specification of payments for each agent.

2.1.2 Payments and Quasi-Linear Utility

As seen in the example, the set of possible decisions $\mathcal{D}$ may sometimes include a payment that each agent has to make. We refer the settings with and without payments as mechanism design with and without money respectively.

The social choice function f can be separated into an outcome rule (which specifies the part of the decision unrelated to payments) and an optional payment rule (which specifies the payments that each agent makes). Specifically, we assume that there is some set of outcomes Ω, with o denoting an element of this set. For example, in cake cutting, the set of outcomes Ω is equivalent $\mathcal{D}$ (since there are no payments) and consists of the feasible allocations of cake. In the CA setting, the set of outcomes consists of the feasible assignments of items to each agent, whereas the set of possible decisions consists of feasible assignments along with payments charged to each agent. We assume that agents have a *valuation function* $V_i : \Theta_i \times \Omega \to \mathbb{R}$ that specifies their values for elements of Ω.

Definition 2.1.2. An *outcome rule* $g : \Theta \to \Omega$ maps a set of reported types to an outcome, and a *payment rule* $p : \Theta \to \mathbb{R}^n_{\geq 0}$ maps a set of reported types to a payment for each agent.

We use $g_i(\theta)$ to denote the part of the outcome affiliated with agent i when appropriate. In cake cutting, this is the part of the cake given to the agent. In CAs, this is the set of items the agent obtains in the auction. In settings without money, $u_i(\theta_i, d) = V_i(\theta_i, o)$ where o is the outcome chosen by decision d. In settings with money, we make the standard assumption of *quasi-linear* preferences, i.e. $u_i(\theta_i, d) = V_i(\theta_i, o) - p_i$ where o is the outcome associated with decision d and p_i is the payment of agent i specified by decision d.

2.1.3 Properties

We are interested in social choice functions with specific properties. A natural and perhaps the most common property to consider is the total value that agents receive from the decision.

Definition 2.1.3. Given agent types θ and decision d, the *efficiency* or *social welfare* of decision d is defined as:

$$sw(\theta, d) = \sum_{i=1}^{n} V_i(\theta_i, o),$$

where o is the outcome associated with decision d.

We can use this definition to define what it means for social choice function to be efficient.

Definition 2.1.4. A social choice function f is *efficient* if for all $\theta \in \Theta$,

$$g(\theta) \in \arg\max_{d \in \mathcal{D}} sw(\theta, d),$$

where g is the outcome part of the social choice function f.

A weaker property than efficiency that does not require comparison of values across different agents is Pareto-efficiency.

Definition 2.1.5. A social choice function f is *Pareto-efficient* if for all $\theta \in \Theta$, there does not exist an outcome $o \in \Omega$ such that $V_i(\theta_i, o) \geq V_i(\theta_i, g_i(\theta))$ for every agent i and $V_j(\theta_j, o) > V_j(\theta_j, g_j(\theta))$ for some agent j, where g is the outcome part of the social choice function f.

In other words, f always selects an outcome such that it is not possible to make every agent weakly better off and at least one agent strictly better off. Note that the definitions of efficiency and Pareto-efficiency look at the value that agents receive rather than agent utilities. This captures the perspective of a system designer who wants to create maximal value for society. Payments are not considered in these definitions though they affect agents' utilities for the chosen decision.

2.2 Mechanism Design

In the previous section, we assume that we somehow know the agents' true type θ, whether it is because agents are truthful or because their types are publicly known. In this section,

we consider the mechanism design problem, which assumes that this information is private to each agent. We separate our discussion into two separate cases. In the first case, we assume a *mechanism design without money* setting. In other words, we are not able to use payments to incentivize agents to report their valuations truthfully. This is a realistic assumption in certain settings where payments may be inappropriate such as negotiating land allocation in peace treaties or deciding how to allocate assets in a divorce settlement [Brams and Taylor, 1996]. The cake cutting problem typically assumes that payments are not permitted. In the second case, we consider a *mechanism design with money* setting where payments can be used to incentize agents to report truthfully. This setting includes CAs.

In the first case, a *direct mechanism* h consists of just an outcome rule g. In the second case, a *direct mechanism* h is a pair (g, p), where g is an outcome rule, and $p : \Theta \to \mathbb{R}^n_{\geq 0}$ specifies a payment for each agent. A direct mechanism takes on the same form as a social choice function. The reason we separate the terminology is to allow for indirect mechanisms which allow agents to make reports that lie in a space different than Θ_i. Indirect mechanisms are beyond the scope of this dissertation, but we give a brief discussion in relation to the revelation principle in Section 2.2.2.

Since we assume agent types are private information, we seek mechanisms that induce agents to make truthful reports. To unify our discussion of the two settings, we define agent utilities in each setting.

Definition 2.2.1. For a given mechanism h, the utility of an agent i depends on its true type θ_i, its reported type θ'_i, and the reports of other agents θ_{-i}.

For a mechanism without money $h = g, g : \Theta \to \Omega$, the utility of an agent i is simply its value for the chosen outcome:

$$u_i((\theta'_i, \theta_{-i}), \theta_i) = V_i(\theta_i, g(\theta'_i, \theta_{-i}))$$

For a mechanism with money $h = (g, p), g : \Theta \to \Omega, p : \Theta \to \mathbb{R}^n_{\geq 0}$, we make the standard assumption of quasi-linear preferences. The agent's utility is its value for the outcome minus the payment it has to make:

$$u_i((\theta'_i, \theta_{-i}), \theta_i) = V_i(\theta_i, g(\theta'_i, \theta_{-i})) - p_i(\theta'_i, \theta_{-i})$$

Given these definitions, it is possible to quantify the amount that an agent can gain by misreporting its type.

Definition 2.2.2. Suppose that agent i has true type θ_i and the other agents report types θ_{-i}. Agent i's *ex post regret* is defined as:

$$rgt_i(\theta_i, \theta_{-i}) = \max_{\theta_i' \in \Theta_i} u_i\big((\theta_i', \theta_{-i}), \theta_i\big) - u_i\big((\theta_i, \theta_{-i}), \theta_i\big).$$

It is natural to consider mechanisms that do not make agents worse off if we assume that the agents' reported types are their true types.

Definition 2.2.3. A mechanism h is *individually rational* (IR) if agents reporting their true types are guaranteed non-negative utility, i.e., if for all $i \in N$, $\theta_i \in \Theta_i$, and $\theta_{-i} \in \Theta_{-i}$, $u_i((\theta_i, \theta_{-i}), \theta_i) \geq 0$.

The mechanism h plays an implicit role in the definitions for both regret and individual rationality since agent utilities u_i depend on the particular choice of h.

2.2.1 Truthful Mechanisms

We can now define precisely what it means for a mechanism to induce truthful reports.

Definition 2.2.4. A mechanism (without money) $h = g, g : \Theta \to \Omega$ or a mechanism (with money) $h = (g, p), g : \Theta \to \Omega, p : \Theta \to \mathbb{R}^n_{\geq 0}$ is *dominant strategy incentive compatible* (DSIC) if each agent maximizes its utility by reporting its true type, irrespective of the reports of the other agents, i.e., if for all $i \in N$, $\theta_i \in \Theta_i$, $\theta_{-i} \in \Theta_{-i}$, and $\theta_i' \in \Theta_i$,

$$u_i((\theta_i, \theta_{-i}), \theta_i) \geq u_i((\theta_i', \theta_{-i}), \theta_i).$$

We often use incentive compatible or strategyproof to refer to DSIC. If a mechanism is DSIC, agents experience zero ex post regret. In the case where the outcome rule and payment rule are non-deterministic,[2] we require an alternate notion of truthfulness.

Definition 2.2.5. A mechanism (without money) $h = g$ or a mechanism (with money) $h = (g, p)$ is *truthful in expectation*, if each agent maximizes its expected utility by reporting truthfully (regardless of the reports of other agents), where the expectation is taken over the randomness of the mechanism, i.e., if for all $i \in N, \theta_i \in \Theta_i, \theta_{-i} \in \Theta_{-i}, \theta_i' \in \Theta_i$,

$$E[u_i((\theta_i, \theta_{-i}), \theta_i)] \geq E[u_i((\theta_i', \theta_{-i}), \theta_i)].$$

[2] In this case it is more sensible to consider a combined outcome and payment rule that maps to the simplex over $\Omega \times \mathbb{R}^n_{\geq 0}$.

We mostly consider deterministic mechanisms in this dissertation and do not focus on truthfulness in expectation, but we introduce it here to give background for understanding related work. In the situation where agents' types are drawn from some distribution $P(\Theta)$, we can also define a weaker notion of truthfulness.

Definition 2.2.6. Given that agent types are drawn from $P(\Theta)$, a mechanism (without money) $h = g$ or a mechanism (with money) $h = (g, p)$ is *Bayes-Nash incentive compatible* (BIC) if each agent maximizes its expected utility by reporting its true type, assuming that other agents are truthful, i.e., if for all $i \in N$, $\theta_i \in \Theta_i$

$$E_{\theta_{-i} \sim P(\Theta_{-i}|\theta_i)}[u_i((\theta_i, \theta_{-i}), \theta_i)] \geq E_{\theta_{-i} \sim P(\Theta_{-i}|\theta_i)}[u_i((\theta'_i, \theta_{-i}), \theta_i)].$$

BIC is a weaker notion than DSIC since BIC only requires that reporting the true type θ_i is weakly best in expectation. It could be that there are some types of other agents θ_{-i} for which θ_i is not the best report, but θ_i yields weakly better expected utility over the random draw of θ_{-i}. Unlike DSIC mechanisms where agents have zero ex post regret, agents may experience positive ex post regret in truthful in expectation and BIC mechanisms depending on the particular random choice of the mechanism or random draw of agents' types. However, in expectation (and assuming other agents are truthful in the case of BIC), an agent maximizes its utility by reporting its true type.

A Characterization of Strategyproof Mechanisms

Having provided the definition of a DSIC mechanism, it is possible to give a characterization of the outcome rule and payment rule pairs (for mechanism design with money settings) that yield DSIC mechanisms. For the purposes of this dissertation, we consider settings with natural restrictions on the agents' valuations and the outcome rules g.

Definition 2.2.7. Agent valuations exhibit *no externalities* if an agent's value for an outcome only depends on its part of the outcome, i.e. $V_i(\theta_i, o) = V_i(\theta_i, o_i)$.

Definition 2.2.8. Outcome rule g satisfies *consumer sovereignty* if for all $i \in N$, $o_i \in \Omega_i$, and $\theta'_{-i} \in \Theta_{-i}$, there exists $\theta'_i \in \Theta_i$ such that $g_i(\theta'_i, \theta'_{-i}) = o_i$.

In other words for every possible outcome, there exists some reported type for agent i that causes the mechanism to select that outcome. For example, in a single-item auction that assigns the item to the highest bidder, if an agent submits a high enough bid, the agent will win the item.

Definition 2.2.9. Outcome rule g satisfies *reachability of the null outcome* if for all $i \in N$, $\theta_i \in \Theta_i$, and $\theta'_{-i} \in \Theta_{-i}$, there exists $\theta'_i \in \Theta_i$ such that $v_i(\theta_i, g_i(\theta'_i, \theta'_{-i})) = 0$.

Reachability of the null outcome requires that there always exists some outcome that gives the agent zero utility, and the agent can always cause this outcome to be selected. In the single-item auction example, an agent can always receive zero utility by bidding zero. The agent will not receive the item and pays zero and ends up with utility zero.[3]

Given the definitions of DSIC mechanisms and individual rationality from the previous sections, observe that a DSIC mechanism with an outcome rule that satisfies reachability of the null outcome implies individual rationality. The following characterization of DSIC mechanisms is well-known (see e.g., Proposition 9.27 in Nisan [2007]) and easy to prove. It is crucial to the discussion in Chapter 9 which draws connections between mechanism design and classification problems in machine learning. We adapt it here for the case where consumer soveriegnty holds.

Theorem 2.2.10. *A mechanism with money (g, p) that satisfies consumer sovereignty is DSIC if and only if the payment of an agent is independent of its reported type and the chosen outcome simultaneously maximizes the utility of all agents, i.e., if for every type profile $\theta \in \Theta$,*

$$p_i(\theta) = t_i(\theta_{-i}, g_i(\theta)) \qquad \text{for all } i \in N, \text{ and} \tag{2.1}$$

$$g_i(\theta) \in \arg\max_{o'_i \in \Omega_i} \big(v_i(\theta_i, o'_i) - t_i(\theta_{-i}, o'_i)\big) \qquad \text{for all } i \in N, \tag{2.2}$$

for a price function $t_i : \Theta_{-i} \times \Omega_i \to \mathbb{R}$.

The first property is the *agent-independent* property: conditioned on reports of others, and an outcome, an agent's payment is independent of its own report. The second property is the *agent-optimizing* property: the outcome should maximize an agent's utility given these agent-independent prices and its reported valuation.

DSIC mechanisms can also be characterized in regard to necessary and sufficient properties of outcome rules alone, and especially through *monotonicity* properties. These properties characterize those outcome rules for which there exists a payment rule such that the outcome rule and payment rule form a DSIC mechanism [Saks and Yu, 2005, Ashlagi et al., 2010].

[3] If all other agents bid zero there might be some chance the agent wins the item and gains positive utility. To preclude this case we can let the agent bid a negative amount.

A simple necessary condition for DSIC mechanisms is weak monotonicity [Lavi et al., 2003, Bikhchandani et al., 2006].

Definition 2.2.11. An outcome rule g is *weakly monotone* if for all $\theta_i, \theta_i' \in \Theta_i$, $\theta_{-i} \in \Theta_{-i}$,

$$V_i(\theta_i', g(\theta_i', \theta_{-i})) - V_i(\theta_i', g(\theta_i, \theta_{-i})) \geq V_i(\theta_i, g(\theta_i', \theta_{-i})) - V_i(\theta_i, g(\theta_i, \theta_{-i}))$$

In other words, the difference in value for between outcomes $g(\theta_i', \theta_{-i})$ and $g(\theta_i, \theta_{-i})$ is weakly greater when an agent has type θ_i' versus type θ_i.

Depending on the domain of permitted agent valuations (i.e., the definitions of the sets Θ_i), weak monotonicity may also be a sufficient condition for the existence of a payment rule p such that (g, p) is DSIC. A full discussion of this is beyond the scope of this dissertation and the interested reader can consult [Lavi et al., 2003, Bikhchandani et al., 2006] and references therein.

2.2.2 Truthfulness and the Revelation Principle

We have given definitions for truthful mechanisms. In this section, we provide some motivation for the study of truthful mechanisms. One advantage of truthful mechanisms is that is it easy for agents to participate in a truthful mechanism. In a DSIC or truthful in expectation mechanism, agents need need not reason about possible reports that other agents are making. Related to this, a DSIC or truthful in expectation mechanism is fair in the sense that unsophisticated agents are not at a disadvantage relative to more sophisticated, strategic agents or agents with more information about other agents' types. Additionally, truthful mechanisms are more predictable from the perspective of the mechanism designer. If we depart from DSIC or truthful in expectation mechanisms, we need to think about what deviations agents will make and how agents will respond to these deviations. In other words, we need to consider equilibrium behavior. BIC mechanisms also depend on equilibrium reasoning (the expected utility is taken over the true type distribution), but we can argue that this truthful equilibrium is more likely than other more complicated equilibria given that it consists of the straightforward strategy of reporting one's true type. A final reason for the study of truthful mechanisms is that they are in some sense without loss due to what is known as the revelation principle.

Before we discuss the revelation principle, we consider mechanisms that are not truthful and how we can think about analyzing such mechanisms. The following section is specifically focused on the application of game theory to mechanism design, but we introduce several

general game theory concepts such as strategies and equilibrium. Mas-Colell et al. [1995] and Nisan et al. [2007] offer more comprehensive treatments.

Strategies, Indirect Mechanisms, and Solution Concepts

While we have introduced direct, truthful mechanisms, we can also consider non-truthful, indirect mechanisms. An indirect mechanism asks for agents to report some message $m_i \in \mathcal{M}_i$, where $\mathcal{M}_i$ is not necessarily the same as Θ_i. It then takes the messages and computes an outcome and optionally payments for each agent. We use the notation $u_i((m_i, m_{-i}), \theta_i)$ to denote an agent's utility in an indirect mechanism when it has type θ_i, it reports m_{-i}, and other agents report m_{-i}.

The first step in thinking about non-truthful mechanisms is to realize that agents will now have strategies. A strategy dictates the actions an agent will take in different states of the world. In the context of indirect mechanisms, a strategy s_i indicates for every type θ_i, the message $s_i(\theta_i) = m_i$ the agent will report to the mechanism.[4] We would like to make predictions about what kinds of strategies will arise when we deploy our non-truthful mechanism. This brings us to the concept of equilibrium, i.e. strategies that are mutual best responses to each other. We can now discuss increasingly general notions of equilibrium in mechanisms. We refer to these different notions of equilibrium as *solution concepts*.

Definition 2.2.12. A set of strategies $(s_1, \ldots, s_n)$ is a *dominant-strategy equilibrium* of a mechanism if for all i, for all $\theta_i \in \Theta_i$, for all $m_i \in \mathcal{M}_i$, for all $m_{-i} \in \mathcal{M}_{-i}$,

$$u_i((s_i(\theta_i), m_{-i}), \theta_i) \geq u_i((m_i, m_{-i}), \theta_i).$$

Dominant-strategy equilibrium is very strong. It requires that for every possible type θ_i of an agent, there exists some report $s_i(\theta_i)$ that dominates all other reports, regardless of the reports of other agents. We also have a weaker notion when agent types are drawn from a distribution $P(\Theta)$.

Definition 2.2.13. Assuming that agent types are drawn from $P(\Theta)$, a set of strategies $(s_1, \ldots, s_n)$ is a *Bayes-Nash equilibrium* of a mechanism if for all i, for all $\theta_i \in \Theta_i$, $m_i \in \mathcal{M}_i$,

$$E_{\theta_{-i} \sim P(\Theta_{-i}|\theta_i)}[u_i((s_i(\theta_i), s_{-i}(\theta_{-i})), \theta_i)] \geq E_{\theta_{-i} \sim P(\Theta_{-i}|\theta_i)}[u_i((m_i, s_{-i}(\theta_{-i})), \theta_i)].$$

[4] Technically we can also allow randomization so that s_i maps Θ to $\Delta\mathcal{M}_i$, but this distinction is not important for the purposes of our discussion.

Recalling our definitions of DSIC and BIC mechanisms from the previous section, these equilibrium definitions look very similar, except that we have introduced strategies $s_i(\theta_i)$ to replace the agents' true type θ_i and we have generalized the space of possible reports to be $\mathcal{M}_i$ rather than Θ_i. Indeed, the revelation principle makes this relationship precise and basically says that any mechanism with a dominant strategy or Bayes-Nash equilibrium can be converted to a DSIC or BIC mechanism.

When agents strategize, from the system design perspective, we are interested in the decision (either the outcome or the outcome-payment pair) that is chosen with respect to the agents' true types. In other words, we are interested in the social choice function that results from running a mechanism.

Definition 2.2.14. Given a set of strategies $(s_1, \ldots, s_n)$ that is a dominant strategy or Bayes-Nash equilibrium of a mechanism h, we can defined the affiliated social choice function f that takes into takes into account the strategies, i.e. $f(\theta) = h(s(\theta))$. In this case, we say that $(s_1, \ldots, s_n)$ *implements* f in {dominant strategies, Bayes-Nash equilibrium}.

We are now ready to give the revelation principle.

Theorem 2.2.15. *If some set of strategies $s_1, \ldots, s_m$ and possibly indirect mechanism h implement a social choice function f in dominant strategies or Bayes-Nash equilibrium, then there exists a direct truthful mechanism that implements f in dominant strategies or Bayes-Nash equilibrium respectively.*

We do not give a formal proof of the theorem, but the intuition is that given some indirect mechanism with an equilibrium, we can consider a direct mechanism that plays the equilibrium on behalf of the agents and feeds this into the indirect mechanism. The equilibrium conditions then guarantee that truthful reporting is the agent's best strategy in the constructed direct mechanism.

The revelation principle was discovered for dominant strategies by Gibbard [1973] and extended to Bayes-Nash equilibria by Dasgupta et al. [1979], Holmström [1979], and Myerson [1979]. The principle states that if we have some non-truthful indirect mechanism that implements a social choice function f, then there exists a truthful, direct mechanism that also implements f (if our solution concepts are dominant strategy equilibrium or Bayes-Nash equilibrium). Therefore, in some sense, it is without loss of generality to focus our study on truthful mechanisms. Indeed, it is much simpler to reason about truthful mechanisms than to think about indirect mechanisms which can allow agents to make very complicated re-

ports to the mechanism, and in addition, among direct mechanisms, the revelation principle says that we can focus on truthful mechanisms.

On the other hand, the revelation principle does not have anything to say about the particular details of a mechanism. It states that there is a truthful, direct mechanism that implements the same social choice function f, but an indirect mechanism may implement the same function with far less communication and/or computation. Indeed, communication and computation become very relevant for settings like CAs and it may be beneficial to use indirect mechanisms (see e.g., Parkes [2001]).

2.2.3 Classic Mechanism Design Results

Having presented the mechanism design problem and provided some motivation for studying truthful mechanisms, we now give some fundamental results from the mechanism design with money literature.

Vickrey-Clarke-Groves Mechanisms

Definition 2.2.16. Suppose the agents' reported types are θ. A Vickrey-Clarke-Groves (VCG) mechanism [Vickrey, 1961, Clarke, 1971, Groves, 1973] is a direct mechanism that consists of the following:

1. An outcome rule g that selects an outcome $o \in \Omega$ that maximizes social welfare with respect to the reported types θ.

2. A payment rule p_i that charges

$$p_i = t_i(\theta_{-i}) - \sum_{j \neq i} V_j(\theta_j, g_j(\theta)),$$

   where t_i is any function that depends only on θ_{-i}.

VCG mechanisms define an entire family of mechanisms since the choice of t_i is left open. VCG mechanisms are of great interest as they are DSIC.

Theorem 2.2.17. *VCG mechanisms are DSIC.*

Proof. Consider an agent's utility when reporting type θ_i' when its true type is θ_i:

$$U_i((\theta_i', \theta_{-i}), \theta_i) = V_i(\theta_i, g(\theta_i', \theta_{-i})) - p_i(\theta_i', \theta_{-i}) \tag{2.3}$$

$$= V_i(\theta_i, g(\theta_i', \theta_{-i})) - \left(t_i(\theta_{-i}) - \sum_{j \neq i} V_j(\theta_j, g(\theta_i', \theta_{-i})) \right) \tag{2.4}$$

$$= \sum_{j=1}^{n} V_j(\theta_j, g(\theta_i', \theta_{-i})) - t_i(\theta_{-i}). \tag{2.5}$$

Agent i has no effect on the term $t_i(\theta_{-i})$, so we can focus on the first term. g maximizes social welfare, so if $\theta_i' = \theta_i$, then the outcome o^* chosen will maximize $\sum_{j=1}^{n} V_j(\theta_j, o)$. If the agent reports $\theta_i' \neq \theta_i$, then g will choose an outcome that maximizes social welfare with respect to (θ_i', θ_{-i}), and this outcome may not maximize $\sum_{j=1}^{n} V_j(\theta_j, o)$. Therefore, agent i is weakly better off by reporting its true type θ_i. □

Not only are VCG mechanisms DSIC, they are the only efficient, DSIC mechanisms if the space of possible valuations satisfies a connectedness property [Green and Laffont, 1973, Holmström, 1979]. A common choice for the function t_i is the Clarke-Pivot rule, which sets

$$t_i = \max_{o \in \Omega} \sum_{j \neq i} V_j(\theta_j, o).$$

The appeal of this particular choice for t_i is that the resulting mechanism is individually rational for many natural settings including combinatorial auctions without externalities (i.e. each agent's value for an outcome only depends on the items it receives).

Single-Parameter Mechanism Design

VCG mechanisms are very general and apply to many different settings as long as the outcome rule maximizes social welfare. In this section, we discuss mechanism design in the special setting where agents' valuations can be summarized by a single real number. In this setting, we have very nice characterizations of incentive-compatible mechanisms. The following discussion is adapted from [Nisan et al., 2007].

In a single-parameter mechanism design problem, each agent's type is a single real number θ_i that lies in some interval $\Theta_i = [\ell_i, u_i], \ell_i \geq 0$, and each agent has a publicly known set of *winning outcomes* $\Omega_i^* \subseteq \Omega$. The agent's value for a winning outcome is θ_i and 0 for any other outcome. As an example, we can consider a single-item auction. Here the single-parameter is an agent's value for the item, and the winning outcomes are the outcomes

that allocate the item to the agent. A more complex example of a single-parameter setting is a CA where each agent is interested in exactly one publicly known target bundle of items (but their value for that bundle is not known). This is what is called a known single-minded CA and is discussed further in Chapter 7. Here Ω_i^* consists of all allocations that give the agent any superset of its target bundle.

A mechanism is *normalized* if an agent pays 0 when it is not winning. We have an exact characterization of normalized incentive compatible mechanisms.

Theorem 2.2.18 ([Myerson, 1981])**.** *A normalized mechanism (g, p) is incentive compatible iff*

1. *For all $\theta_{-i} \in \Theta_{-i}$, $\theta_i, \theta_i' \in \Theta_i, \theta_i < \theta_i'$, $g_i(\theta_i, \theta_{-i}) \in \Omega_i^* \Rightarrow g_i(\theta_i', \theta_{-i}) \in \Omega_i^*$. In other words, if non-winning outcomes are mapped to 0 and winning outcomes are mapped to 1, then $g_i(\cdot, \theta_{-i})$ should be monotone.*

2. *$p_i(\theta_i, \theta_{-i})$ is defined as*

   (a) 0 if $g_i(\theta_i, \theta_{-i}) \notin \Omega_i^*$,

   (b) $\sup\{\theta_i' \in \Theta_i : g_i(\theta_i', \theta_{-i}) \notin \Omega_i^*\}$ if there exists some $\theta_i' \in \Theta_i$ where $g_i(\theta_i', \theta_{-i}) \notin \Omega_i^*$,

   (c) $q_i(\theta_{-i})$ otherwise.

The payment rule part of the Theorem 2.2.18 requires that payments to non-winning agents are 0 and that we charge winning agents the *threshold value* at which they start winning. If an agent always wins when other agents report θ_{-i}, then we just require that the payment be some value that does not depend on the agent's report.

Theorem 2.2.18 reduces the problem of finding truthful mechanisms to the problem of finding monotone outcome rules. Once we have a monotone outcome rule, the payment rule is pinned down by finding the threshold at which an agent goes from non-winning to winning. The theorem can be generalized to randomized mechanisms and BIC mechanisms for single-parameter settings. We refer the interested reader to [Myerson, 1981] or [Hartline and Karlin, 2007] for more details.

Chapter 3

Cake Cutting

Cutting a cake is often used as a metaphor for allocating a divisible good. The difficulty is not cutting the cake into pieces of equal size, but rather that the cake is not uniformly tasty: different agents prefer different parts of the cake, depending, e.g., on whether the toppings are strawberries or cookies. The goal is to divide the cake in a way that is "fair"; the definition of fairness is a nontrivial issue in itself, which we discuss in the sequel.

The cake cutting problem dates back to the 1940s [Steinhaus, 1948] and for over sixty years has attracted the attention of mathematicians, economists, and political scientists. While most of the work in artificial intelligence, and computer science in general, has focused on the allocation of indivisible resources, recent years have seen an increasing interest among computer scientists in the allocation of divisible resources ([Edmonds and Pruhs, 2006b,a, Procaccia, 2009, Chen et al., 2010, Zivan et al., 2010, Maya and Nisan, 2012, Kurokawa et al., 2013, Brânzei and Miltersen, 2013, Brânzei et al., 2013]).

On the one hand, the allocation of divisible resources is of relevance to multi-agent resource allocation [Chevaleyre et al., 2006]. On the other hand, computer science brings an interesting perspective to the study of cake cutting related to thinking about the computational complexity of cake cutting and the representation and communication of agents' valuations. In this chapter, we formally define the cake cutting problem and the necessary foundation to understand our contributions to the cake cutting literature.

3.1 Model

We consider a heterogeneous cake, represented by the interval $[0, 1]$. A *piece of cake* is a finite union of subintervals of $[0, 1]$. We sometimes abuse this terminology by treating a piece of cake as the set of the (inclusion-maximal) intervals that it contains. The length

of the interval $I = [x, y]$, denoted $\text{len}(I)$, is $y - x$. For a piece of cake X we denote $\text{len}(X) = \sum_{I \in X} \text{len}(I)$.

The set of agents is denoted $N = \{1, \ldots, n\}$. Each agent $i \in N$ holds a private valuation function V_i, which maps given pieces of cake to the value agent i assigns them. Formally, each agent i has a *value density function*, $v_i : [0, 1] \to [0, \infty)$, that is piecewise continuous. The function v_i characterizes how agent i assigns value to different parts of the cake. The value of a piece of cake X to agent i is then defined as $V_i(X) = \int_X v_i(x)dx = \sum_{I \in X} \int_I v_i(x)dx$. We note that the valuation functions are *additive*, i.e. for any two disjoint pieces X and Y, $V_i(X \cup Y) = V_i(X) + V_i(Y)$, and *non-atomic*, that is $V_i([x, x]) = 0$ for every $x \in [0, 1]$. The last property implies that we do not have to worry about the boundaries of intervals, i.e., open and closed intervals are identical for our purposes. While some of the cake-cutting literature assumes that valuations are *absolutely continuous* (see e.g., Brams et al. 2012b), i.e., that if any agent attaches zero value to a portion of the cake, then all other players do, we do not make this assumption.

The output of a cake cutting algorithm is an *allocation* $A_1, \ldots, A_n$ of pieces of cake to each agent such that pieces A_i are pairwise disjoint. For each $i \in N$ the piece A_i is allocated to agent i, and the rest of the cake, i.e., $[0, 1] \setminus \bigcup_{i \in N} A_i$, is thrown away. We assume *free disposal*, i.e., it is not necessary for the algorithm to allocate the entire cake and resources can be thrown away without incurring a cost.

3.2 Fairness

Various notions of fairness have been studied in the literature. Here we introduce the two most prominent notations: proportionality and envy-freeness. An allocation $A_1, \ldots, A_n$ is *proportional* if for every $i \in N$, $V_i(A_i) \geq V_i([0, 1])/n$, that is, each agent receives at least a $(1/n)$-fraction of the cake according to its own valuation. An allocation is *envy-free (EF)* if for every $i, j \in N$, $V_i(A_i) \geq V_i(A_j)$, i.e., each agent prefers its own piece of cake to the piece of cake allocated to any other agent. A proportional (resp., EF) cake cutting algorithm always returns a proportional (resp., EF) allocation.

Note that when $n = 2$ proportionality implies envy-freeness. Indeed, $V_i(A_i) + V_i(A_{3-i}) \leq 1$, and hence if $V_i(A_i) \geq 1/2$ then $V_i(A_{3-i}) \leq 1/2$. Under the free disposal assumption the converse is not true. For example, an allocation that throws away the entire cake is EF but not proportional. In general, when $n > 2$ proportionality neither implies nor is implied by envy-freeness. If free disposal is not assumed, that is, the entire cake is allocated, then

envy-freeness implies proportionality for any n.

A third, less commonly studied notion of fairness is *equitability* (EQ). An allocation is equitable if $V_i(A_i) = V_j(A_j)$ for all pairs of agents i, j.

3.3 Normalization of Valuations

Throughout our discussion, we assume that agent valuations are normalized so that $V_i([0,1]) = 1$ for every agent i, i.e., we divide each agents' density function by $V_i([0,1])$. This assumption is without loss if we are looking at proportional and/or EF allocations. An allocation is proportional and/or EF for normalized valuations if and only if it is also proportional and/or EF when valuations are not normalized. Similarly, the assumption is without loss if we consider notions such as Pareto-efficiency. On the other hand, if we consider optimizing social welfare or the fairness notion of EQ, then normalizing valuations can change the allocations we consider to be fair or welfare maximal. In Chapter 4, we look at maximizing social welfare and EQ, so the assumption is not without loss. However, our algorithmic results extend to the unnormalized case as well, and our impossibilities are only strengthened by widening the possible valuations. In Chapter 5 and 6, we discuss only fairness criteria for which the assumption is without loss of generality.

3.4 Models of Interaction

The existence of proportional, EF, and EQ allocations has been well-known since the 1940s (see Brams and Taylor [1995]). The cake cutting challenge is to come up with procedures that find such allocations. Prior work focuses on algorithms that accommodate arbitrarily complex valuation functions. One of our innovations is advocating the study of cake cutting under a direct revelation model that restricts consideration to cases when valuation functions have a succinct representation.

3.4.1 Classic Model

Most of the literature on cake cutting looks for procedures that interact with the agents and make decisions based on these interactions. Agents' complete valuations are never fully revealed, but the procedures guarantee that a fair division will be found if agents follow the procedure truthfully. A *discrete* cake cutting procedure is a procedure where there are a discrete number of interactions with the agents, with later interactions possibly conditioned

on earlier interactions. We are purposely vague about what constitutes an interaction here, but this will become clearer later in this section. In contrast to a discrete procedure, a *moving knife* algorithm allows an impartial referee to move a knife continuously across the cake, stopping along the way when certain conditions are met (e.g., when an agent's value for the part to the left of the cake exceeds a certain value). The key difference is that moving knives require this continuous maneuver of the knife, and there are some moving knife procedures that cannot be simulated using a discrete procedure. We survey some of the classic cake cutting results below.

Cut and Choose

Perhaps the most well-known cake cutting procedure is the Cut and Choose procedure for two agents. In Cut and Choose, the first agent cuts the cake at a point a such that it values $[0, a]$ at $1/2$ and $[a, 1]$ at $1/2$. The second agent chooses the piece that it prefers. This is a discrete procedure as there are just two steps in the algorithm. It is easy to see that this procedure is both proportional and EF. For proportionality, both agents receive value at least $1/2$. For EF, agent 1 is indifferent between the two pieces and agent 2 receives the piece it prefers.

Dubins-Spanier and Banach-Knaster

In order to obtain proportionality for n agents, we need a more clever procedure. We start by examining the moving knife procedure of Dubins and Spanier [1961]. An impartial referee moves a knife starting from 0 to the right. Whenever the piece to the left of the knife is worth value $1/n$ to an agent, the agent shouts 'stop' and is given the piece to the left of the knife. We then restart the procedure with the remaining cake.

This moving knife procedure results in a proportional allocation. It is clear that the agent that shouts 'stop' receives value at least $1/n$. The key observation is that for the remaining agents, the rest of the cake is worth at least value $(n-1)/n$. We can make an inductive argument and argue that $n-1$ agents will yell 'stop' (and therefore receive value exactly $1/n$) and that the last agent remaining will have value at least $1/n$ for the remaining cake.

While this procedure uses a moving knife, it can be converted to a discrete cake cutting procedure that does not require an impartial referee to continuously move the knife across the cake. We can ask each agent to cut a piece starting at 0 and moving right that is worth value exactly $1/n$. We then give the agent with the leftmost cut the piece starting from 0

and ending at its cutpoint. We can then repeat the procedure for the rest of the cake. This is essentially the procedure discovered by Banach and Knaster circa 1944 (see e.g., [Brams and Taylor, 1996]).

Though this procedure gives a proportional allocation, the allocation may not be EF. Indeed, though agent 1 receives a piece worth exactly $1/n$, there is no guarantee that the pieces cut by the remaining agents will have value at most $1/n$. In the extreme case, suppose that agent 1 uniformly values $[0, 1]$ but the other agents only receive value from $[(n-1)/n, 1.0]$. Agent 1 will be very envious of the second agent who yells stop since that agent will receive at least the interval $[1/n, (n-1)/n]$ which is a worth a lot more to agent 1 than $[0, 1/n]$ if $n > 2$. Indeed, finding EF allocations proves to be significantly more challenging than finding proportional allocations.

Selfridge-Conway

An EF procedure for three agents was discovered around 1960 independently by John L. Selfridge and John H. Conway [Brams and Taylor, 1996]. The clever procedure proceeds in a number of steps. We label the players as 1, 2, and 3.

1. Initial division.
   (a) Player 1 cuts the cake into three pieces A, B, C that have equal value from its perspective.
   (b) Player 2 takes the piece that it prefers the most and splits it into two pieces such that one of the two pieces has the same value as the second most preferred piece. Wlog assume that the piece preferred most by player 2 is A and the second most preferred piece is B. Player 2 splits A into A' and A'' such that $V_2(A') = V_2(B)$.
   (c) Player 3 chooses the piece it prefers most from among A', B, C.
   (d) If player 3 did not choose A', player 2 chooses A'. Otherwise, player 2 chooses its more preferred piece from B, C.
   (e) Player 1 is given the remaining piece of A', B, C.
2. Division of the trimmings (A'').
   (a) Either player 2 or player 3 receives A'. Let the player that receives A' be denoted by p and the other player by p'.
   (b) p' divides A'' into three equal pieces (according to its valuation).

(c) p chooses its most preferred piece out of these three.

(d) Player 1 chooses its most preferred piece out of the remaining two pieces.

(e) p' is allocated the remaining piece.

We now show that this division is indeed EF. First, consider player 1. Player p receives the trimmed piece A' along with part of the trimmings A''. Player 1 receives either B or C, which from its perspective has value equal to $A' \cup A''$. As a result, player 1 will not envy player p. Player p' receives either B or C along with part of A''. But player 1 prefers its share of A'' to that of player p' (player 1 goes before player 2 when dividing A'') and views B and C equally. So player 1 does not envy p' either.

Consider player 2. Player 2 does not envy anyone after the initial division. If player 2 is p', then player 2 does not envy anyone when viewing the division of A'' in isolation; after all, A'' is divided so that the three pieces are equal according to p'. If player 2 is p, then player 2 also does not envy anyone when viewing the division of A'' in isolation; after all, player p goes first in choosing the division of A''. Therefore, player 2 has no envy since it has no envy after the initial division and no envy for the division of A'' (and valuations are additive).

Consider player 3. Player 3 also does not envy anyone after the initial division since it goes first. Player 3 does not envy the division of A'' based on the same argument as for player 2. Therefore, player 3 also has no envy.

This procedure is quite clever and seems quite delicately constructed so that the ultimate division is EF. Indeed, an EF procedure for $n > 3$ agents is significantly more complex.

Brams and Taylor

Brams and Taylor [1995] made a breakthrough when they introduced a discrete procedure that finds an EF allocation for any number of agents in a finite number of steps. We do not provide a detailed outline of the procedure here (the procedure for just four agents consists of 20 steps), but the procedure operates by creating EF allocations for a fraction of the cake (by having the agents trim pieces to introduce ties as in the Selfridge-Conway procedure) and then recursively handling the leftover parts of the cake. Though the procedure is finite, it is not bounded. For any number of steps T, there exist some valuations of agents for which the procedure takes more than T steps to finish. Saberi and Wang [2009] introduce a moving knife procedure that terminates in a bounded number of steps for 5 agents, but it is not clear if this procedure can be simulated by some discrete procedure. For $n > 5$, we

do not have any bounded EF algorithms that allocate the entire cake. (We can always just throw away the entire cake if we only insist on envy-freeness but allow some parts of the cake to be thrown away.)

3.4.2 Complexity of Cake Cutting

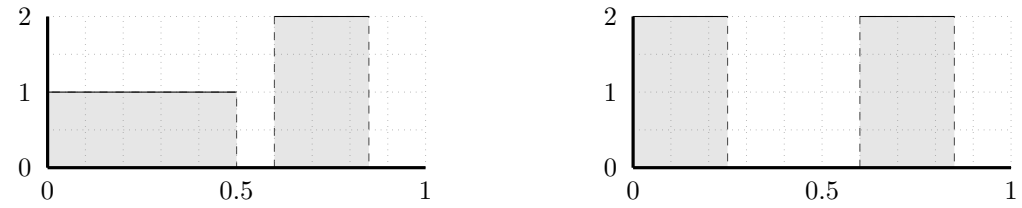
While we have informally defined a discrete cake cutting procedure as a procedure that has a discrete number of steps (in contrast to a moving knife procedure which requires continuously moving a knife), a natural question to consider is how computationally complex is it to find different fair allocations. In order to answer this question, we need to be precise about how we measure complexity.

Robertson and Webb [1998] introduced a *concrete model of complexity* for cake cutting algorithms; under their model an algorithm is restricted to making two types of queries: an evaluation query (whereby the algorithm learns the value of an agent with respect to a given interval) and a cut query (whereby the algorithm obtains a piece worth a given value to an agent). Through the queries the algorithm must obtain sufficient information to output a fair allocation. This model is very general and can capture all the well known discrete cake cutting procedures. For example, the Cut and Choose procedure can be executed by asking the first agent to cut a piece, starting at 0 and moving right, that is worth value exactly 1/2. We can then issue an eval query for one of the pieces to the second agent and give the second agent the piece that is worth at least 1/2.

Under the Robertson and Webb model, proportional cake cutting is well understood. Even and Paz [1984] provide a divide-and-conquer procedure that uses $O(n \log n)$ queries, and lower bound of $\Omega(n \log n)$ was established by Edmonds and Pruhs [2006a]. On the other hand, there remains a large gap in the understanding of EF procedures. Procaccia [2009] establishes a lower bound of $\Omega(n^2)$, but as mentioned above, there are no bounded procedures known for $n > 5$ agents. Recently, Kurokawa et al. [2013] showed that the difficulty of finding bounded procedures persists even if we consider the restricted family of piecewise uniform valuations (introduced below in Section 3.5) as any bounded procedure for piecewise uniform valuations would imply a bounded procedure for general valuations.

3.4.3 Direct Revelation Model

In this disseration, we depart from Robertson and Webb model and instead assume that the agents report their full valuation functions to the algorithm. In order for this to be reasonable, we restrict agents' valuations to families that allow for succinct representations.



(a) Value density function for a piecewise constant valuation that is not piecewise uniform.

(b) Value density function for a piecewise uniform valuation.

Figure 3.1: An illustration of special value density functions.

We discuss these families of valuation functions in the next section.

A side effect of work in the direct revelation model is that our algorithms are centralized whereas many of the classic algorithms can be implemented in a decentralized way (e.g., Cut and Choose). In particular, an interesting question that we leave to future work is whether the algorithms we present can be efficiently implemented via evaluation and cut queries.

3.5 Families of Valuation Functions

A valuation function V_i is *piecewise linear* if its corresponding value density function v_i is piecewise linear, that is $[0, 1]$ can be partitioned into a finite number of intervals such that v_i is linear on each interval. Similarly, a valuation function V_i is *piecewise constant* if its corresponding value density function v_i is piecewise constant (see Figure 3.1(a)). A valuation function V_i is *piecewise uniform* if moreover v_i is either some constant $c \in \mathbb{R}_+$ (the same one across intervals) or zero. See Figure 3.1(b) for an illustration.

Piecewise uniform valuation functions imply that each agent $i \in N$ is uniformly interested in a finite union of intervals, which we call its *reference piece of cake* and denote by U_i. For example, in Figure 3.1(b), $U_i = [0, 0.25] \cup [0.6, 0.85]$. Given a piece of cake X, it holds that $V_i(X) = \text{len}(X \cap U_i)/\text{len}(U_i)$. Though simple, piecewise uniform valuations capture some natural settings. As an example, suppose that the cake represents access time to a backup server. Each agent is equally interested in time intervals when its computer is idle or when it is not modifying its data.

Piecewise constant and piecewise linear valuation functions are significantly more general than piecewise uniform valuations. Though they can both be succinctly communicated (see the next section), they can both approximate general valuations by using a large number of intervals and selecting a constant or linear density on each interval that approximates the

general value density function.

3.5.1 Computational Complexity

In the sequel, we will discuss various cake cutting algorithms under the direct revelation model and for different families of valuation functions. We are not operating in the Robertson and Webb model, so it will not make sense to measure complexity by the number of queries we make to the agents. Instead, we would like the algorithm that computes the allocation based on agents' reported preferences to take time that is polynomial in the size of the input representation. Therefore, in order to discuss computational complexity (and how computation time scales with the size of the input), we need to clearly define the size of the input to the algorithm.

In all cases, the size of the input is the number of bits needed to specify the agents' preferences. For piecewise uniform valuations, this involves telling the algorithm the endpoints of the desired intervals. For piecewise constant valuations, this involves communicating the endpoints of the desired intervals as well as the value of the density function on each of these intervals. For piecewise linear valuations, this involves communicating the endpoints of the desired intervals as well as the slope and intercept of the density on each of these intervals. We assume that the endpoints of intervals as well as any slopes or intercepts can be expressed using k-bit rationals (i.e., a/b where a and b are both k-bit integers). We seek algorithms that run in time polynomial in k.

Chapter 4

Welfare Maximization and Cake Cutting

The classic literature on cake cutting focuses on finding fair allocations. However, depending on the fairness criterion and agents' valuations, there may by multiple fair allocations. A natural way to choose a single allocation from a set of fair allocations is to consider the social welfare of different fair allocations. In this chapter, we provide algorithms for finding welfare-maximizing fair allocations and discuss properties of these allocations.

4.1 Preliminaries

We define the *social welfare* or *efficiency* of an allocation $A = (A_1, \ldots, A_n)$, which we denote $e(A)$, as the sum of agent values for the pieces they are given, i.e.,

$$e(A) = \sum_{i=1}^{n} V_i(X_i).$$

An allocation A is *maxsum* among a set of possible allocations $\mathcal{S}$ if $e(A) = \max_{A' \in \mathcal{S}} e(A')$. An allocation A is *efficient* if it is maxsum among all possible allocations. We will be interested in looking at maxsum fair allocations, where $\mathcal{S}$ is a set of fair allocations (either proportional, EF, EQ, or combinations of the three). We append the notion of fairness to maxsum when we have a specific notion of fairness in mind. For instance, a maxsum EF allocation refers to an allocation that has maximal social welfare among all EF allocations.

As we note in Section 3.3, we assume throughout this chapter that agent values are normalized so that $V_i([0, 1]) = 1$ for every agent i. Since we are discussing maximizing

social welfare, this assumption is not without loss. We discuss in Sections 4.2.5 and 4.3.3 why our results do not crucially depend on this assumption. Even if normalization were required, we could consider a setting where agents' utilities are relative and depend on the fraction of their total perceived value for the good they end up receiving.

We assume a direct revelation model as discussed in Section 3.4.3. In this model, agents report their valuations $V_1, \ldots, V_n$ to the algorithm (by reporting their value density functions $v_1, \ldots, v_n$, and the algorithm makes an allocation based on these reports. These reports reports may differ from the agent's true valuations. We do not consider truthfulness in this chapter, so our algorithms and results can be thought to compute maxsum fair allocations with respect to reported valuations, or one can assume agents are truthful. We ignore this distinction in the sequel and adopt value V_i and value density v_i in describing our algorithms.

4.2 Computing Welfare Maximizing Fair Allocations

In this section, we consider the problem of giving tractable algorithms for computing a maxsum fair allocation. Our notion of tractability is that our algorithms run in time polynomial in the number of bits needed to communicate the agents' value density functions. We focus specifically on maxsum EF allocations, but we will point our where our methods can be easily adapted to other notions of fairness Section 4.2.5. Our goal in this section is therefore:

Given the value density functions, tractably compute a maxsum EF allocation.

In some cases we relax this goal, asking only for approximate efficiency, approximate envy-freeness, or both.

Our presentation of the results progresses through three levels of generality in terms of the supported valuation functions. In Section 4.2.2 we assume that the valuation functions are piecewise constant. We give a polynomial-time algorithm that computes maxsum EF allocations via a simple linear programming approach.

In Section 4.2.3 we deal with piecewise linear valuations, a rather general class of valuation functions that strictly contains the class of piecewise constant valuations. We first provide an algorithm that singles out a maxsum EF allocation for the case of two agents. Unfortunately, we show that in this setting, and even with two agents, no tractable algorithm exists, as in some instances any maxsum EF allocation must be specified using irrational

numbers. We therefore leverage our (intractable) algorithm to produce a tractable algorithm that computes approximately EF allocations for two agents that are as efficient as any maxsum EF allocation. The algorithm runs in time polynomial in $\log 1/\epsilon$, where ϵ specifies the amount of envy permitted. Technically, this is facilitated by a delicate search procedure, which in particular employs a technique of Papadimitriou [1979] for searching over rational numbers.

Section 4.2.4 deals with general valuation functions and any number of agents. We design a tractable algorithm that computes approximately maxsum, approximately EF allocations by approximating the given valuation functions by piecewise constant functions and employing the results of Section 4.2.2. Our algorithm runs in time polynomial in $1/\epsilon$, where ϵ specifies the deviation from optimality as well as the amount of envy that is permitted.

4.2.1 Related Work

Caragiannis et al. [2009] present a framework for quantifying the efficiency loss due to fairness requirements, including envy-freeness, under general valuation functions. Their *price of envy-freeness* is the worst-case ratio between the total utility under an (unconstrained) maxsum allocation, and the total utility under a maxsum EF allocation. Caragiannis et al. provide a lower bound of $\Omega(\sqrt{n})$ and a weak upper bound of $O(n)$ on the price of envy-freeness, where n is the number of agents. Note that an upper bound singles out in every instance (set of valuation functions) an allocation that achieves a certain ratio ($O(n)$ in this case), but makes no claim as to whether the allocation is optimal in terms of social welfare.

Reijnierse and Potters [1998] design a clever but complex algorithm that computes a Pareto-efficient EF allocation, i.e., an EF allocation such that no other allocation (including non-EF allocations) is at least as good for all the agents and better for at least one agent, when agents hold piecewise constant valuations. The core of their algorithm involves mapping the cake cutting problem for piecewise constant valuations to a linear Fisher market. A linear Fisher market is a setting where there is a set of divisible items, and an agent's value for an item is linear in the amount of the item they receive and additive across items. The subintervals on which all agents' value densities are constant are mapped to items in the Fisher market, and each agent is endowed with the same fixed budget. A Walrasian equilibrium in this market (prices for items such that the market clears, agents spend their budgets, and agents only purchase items with maximal utility to price ratio) turns out to correspond to a Pareto-efficient EF allocation. Such an equilibrium can be computed

approximately using the Eisenberg-Gale convex program [Eisenberg and Gale, 1959] or exactly using methods developed by Devanur et al. [2002]. The main contrast with our work is that we focus on finding maxsum EF allocations. It is easy to see that Pareto-efficient EF allocations may not be maxsum EF. Consider a simple case where agent 1 uniformly wants $[0, 0.5]$ and agent 2 uniformly likes the entire cake $[0, 1]$. Giving $[0, 0.25]$ to agent 1 and $[0.25, 1.0]$ to agent 2 is EF and Pareto-efficient, but it is not maxsum EF. Indeed, we can increase the total value obtained by giving $[0, 0.5]$ and $[0.5, 1.0]$ to agent 2. On the other hand, we show in the next section (see Theorem 4.3.7) that even when agents have piecewise constant valuations, maxsum EF allocations may not be Pareto-efficient. Therefore, our work examines a different question than this earlier work.

Reijnierse and Potters ultimately use their algorithm to compute approximately Pareto-efficient EF allocations under general valuations; our approximation approach for general valuations, presented in Section 4.2.4, is inspired by theirs.

Zivan et al. [2010] present a way to find Pareto-efficient EF allocations that reduce untruthful manipulations, also assuming agents hold piecewise constant valuations. As discussed, maxsum EF allocation are different than Pareto-efficient EF allocations. Additionally, we do not examine strategic issues in this paper.

Nuchia and Sen [2001] provide a procedure which starts from an externally given EF allocation and improves its efficiency while maintaining envy-freeness. However, this procedure is not guaranteed to produce an maxsum EF allocation. In Section 4.2.3 we do provide such a guarantee by starting from an efficient allocation and improving its envy-freeness.

For general valuations, computing EF allocations is notoriously difficult (see, e.g., Procaccia [2009]). However, there is a known approach for computing ϵ-EF allocations via Sperner's Lemma [Su, 1999]. If one is only interested in approximate envy-freeness, our approach is comparably simple, but significantly more general as it makes it possible to also optimize social welfare.

4.2.2 Piecewise Constant Valuations

We first consider algorithms for computing maxsum EF allocations for piecewise constant valuations (see Section 3.5). Though these valuations functions are restrictive, they model certain real settings and these results are leveraged in Section 4.2.4 to address general valuations.

The main result of this section is a simple polynomial-time algorithm for finding a maxsum EF allocation when agents have piecewise constant valuations. As discussed in

Algorithm 1

1. Mark the boundaries of the reported intervals of all agents, as well as 0 and 1.
2. Let $\mathcal{J}$ be the set of subintervals of $[0, 1]$ formed by consecutive marks.
3. Solve the following linear program:

$$\max \quad \sum_{i=1}^{n} \sum_{I \in \mathcal{J}} x_{iI} V_i(I) \tag{4.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{iI} \leq 1 \ \forall I \in \mathcal{J} \tag{4.2}$$

$$\sum_{I \in \mathcal{J}} x_{iI} V_i(I) \geq \sum_{I \in \mathcal{J}} x_{jI} V_i(I) \ \forall i, j \in N \tag{4.3}$$

$$x_{iI} \geq 0 \ \forall i \in N, I \in \mathcal{J} \tag{4.4}$$

4. Return an allocation which for all $i \in N$ and $I \in \mathcal{J}$ allocates an x_{iI} fraction of subinterval I to agent i.

Section 3.5.1, we assume that the endpoints are agents' desired intervals are k-bit rationals and that their value density functions take on values that are k-bit rationals.

Our procedure for finding a maxsum EF allocation is formally given as Algorithm 1. Step 1 of the algorithm is illustrated in Figure 4.1. The linear program (LP) in Step 3 has variables x_{iI} for each $i \in N$ and $I \in \mathcal{J}$ (where $\mathcal{J}$ is defined in Step 2), which represent the fraction of interval I given to agent i. Crucially, the value density functions of all agents are constant on each interval $I \in \mathcal{J}$, hence the value of each agent $i \in N$ for a fraction x_{iI} of interval I is $x_{iI} V_i(I)$, and the agent's value for its piece is $\sum_{I \in \mathcal{J}} x_{iI} V_i(I)$. The objective function (4.1) then simply gives the social welfare of the allocation. The first constraint (4.2) ensures that the allocation of each interval in $\mathcal{J}$ is valid, while the second constraint (4.3) is the envy-freeness constraint. We have the following result.

Theorem 4.2.1. *Assume that there are n agents with piecewise constant valuation functions. Then Algorithm 1 computes a maxsum EF allocation in polynomial time.*

Interestingly, setting the variables x_{iI} to $1/n$ for every $i \in N$ and $I \in \mathcal{J}$—allocating to each agent a $1/n$-fraction of each interval in $\mathcal{J}$—produces an allocation where $V_i(X_j) = 1/n$ for every $i, j \in N$; we call the partition $X_1, \ldots, X_n$ with this property a *perfect partition*. The allocation induced by a perfect partition is in particular EF. So, under piecewise constant valuation functions finding an EF allocation is trivial, and computing a maxsum EF allocation only slightly less so.

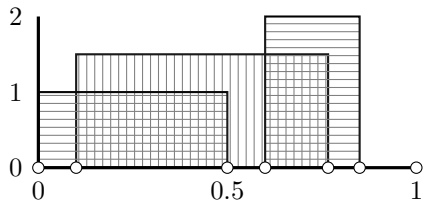


Figure 4.1: An illustration of piecewise contant value density functions, where $n = 2$ and the area under the density function of agent 1 (resp., agent 2) is filled with horizontal (resp., vertical) lines. Marks made by Algorithm 1 are represented by white circles on the horizontal axis. Note that both value density functions are constant between every pair of consecutive marks.

4.2.3 Piecewise Linear Valuations

Piecewise linear valuation functions offer added expressiveness, yet can still be concisely represented. As discussed in Section 3.5.1, the agent can specify the endpoints of the intervals on which its value density function is linear, and then for each interval can provide the slope and intercept of the value density function on that interval.

While Algorithm 1 exactly solves the piecewise constant case, it is not directly generalizable to the piecewise linear case. The algorithm relies on the fact that we can split $[0, 1]$ into a finite number of intervals on which agent value densities were constant. This allows us to focus only on the fraction of each interval given to each agent rather than the specific part of the interval. With piecewise linear valuations, it is not longer possible to split $[0, 1]$ into a finite number of intervals on which value densities are constant.

The main result of this section is an algorithm that finds a maxsum EF allocation for two agents when valuations are piecewise linear. Envy-freeness and proportionality are equivalent in the case of two agents if the entire cake is allocated, so the algorithm equivalently finds a maxsum proportional allocation. We first outline an abstract algorithm for handling these valuation functions. We then prove an impossibility result that an exact implementation of this abstract algorithm is intractable. We conclude by sketching an approximate implementation of the algorithm. An algorithm for any number of agents is left open.

An abstract algorithm

At a high level, the algorithm starts with a maxsum (not necessarily EF) allocation, and transfers pieces to the envious agent until the agent is no longer envious. The crux of the procedure lies in the choice of which pieces are given to the envious agent. A key notion

Algorithm 2

1. If $V_1(Y_{1\geq 2}) \geq 1/2$ and $V_2(Y_{2\geq 1}) \geq 1/2$, give agent 1 $Y_{1>2}$, agent 2 $Y_{2>1}$.
   (a) If $V_1(Y_{1>2}) \geq 1/2$, give $Y_{1=2}$ to agent 2.
   (b) Otherwise, divide $Y_{1=2}$ so that agent 1 receives value exactly $1/2$.
2. Without loss of generality, assume $V_1(Y_{1\geq 2}) < 1/2$. Give $Y_{1\geq 2}$ to agent 1. Let r^* be the maximal r such that $V_1(Y_{1\geq 2} \cup Y_{R_1 \geq r}) \geq 1/2$. Give $Y_{>r^*}$ to agent 1, and divide $Y_{=r^*}$ so that agent 1 receives exactly value $1/2$.

will be that of the ratio between the density functions of agent 1 and agent 2.

Definition 4.2.2. For $x \in [0,1]$ where $v_2(x) \neq 0$, the *value ratio at x for agent 1* is $R_1(x) = v_1(x)/v_2(x)$. For $x \in [0,1]$ where $v_1(x) \neq 0$, the *value ratio at x for agent 2* is $R_2(x) = v_2(x)/v_1(x)$.

Notationally, define the sets

$$Y_{i \ op \ j} = \{x \in [0,1] : \ v_i(x) \ op \ v_j(x)\} \tag{4.5}$$

$$Y_{R_1 op \ r} = \{x : v_1(x) \leq v_2(x), v_2(x) > 0, R_1(x) \ op \ r\} \tag{4.6}$$

$$Y_{R_2 op \ r} = \{x : v_2(x) \leq v_1(x), v_1(x) > 0, R_2(x) \ op \ r\} \tag{4.7}$$

where $i, j \in \{1,2\}$ and $op \in \{>, \geq, =\}$. For instance, $Y_{1\geq 2} = \{x \in [0,1] : \ v_1(x) \geq v_2(x)\}$ and $Y_{R_1 > r} = \{x \in [0,1] : v_1(x) \leq v_2(x), v_2(x) > 0, R_1(x) > r\}$.

Using these notations we can present our algorithm, given as Algorithm 2. In the rest of this subsection we prove the following theorem.

Theorem 4.2.3. *Assume that there are two agents with piecewise linear valuations. Algorithm 2 finds a maxsum EF allocation.*

Before proving Theorem 4.2.3, we establish a few useful lemmas.

Lemma 4.2.4. *Suppose that agent $i \in \{1,2\}$ receives a piece of cake X_i, with $V_i(X_i) \geq 1/2$. Agent i will not envy the other agent.*

Proof. By additivity, $V_i(X_i) + V_i([0,1] \setminus X_i) = 1$. The proposition follows by observing that the other agent receives at most $[0,1] \setminus X_i$ if agent i receives X_i. □

Lemma 4.2.5. *In any maxsum EF allocation, all intervals desired by at least one agent are allocated (i.e., desired intervals are not discarded).*

Proof. Suppose for contradiction that there is some maxsum EF allocation X_1, X_2 that does not allocate an interval I where $v_1(I) > 0$ or $v_2(I) > 0$. We can augment X_1, X_2 with an allocation of I that maintains envy-freeness while improving efficiency. Indeed, assume without loss of generality that there exists on I where $v_1(I) > 0$. Divide I into two subintervals I', I'' such that $V_1(I') = V_1(I'')$. Allocate to agent 2 the subinterval with higher value according to V_2, and give the remaining subinterval to agent 1. Social welfare is improved because agent 1 receives strictly greater value and agent 2 receives weakly greater value. Envy-freeness is maintained since agent 1 is indifferent between the two pieces, and agent 2 prefers the additional piece it receives. □

The following is a simple consequence of this observation.

Lemma 4.2.6. *In any maxsum EF allocation,* $V_1(X_1) \geq 1/2$ *and* $V_2(X_2) \geq 1/2$.

Proof. By Lemma 4.2.5, all desired intervals are allocated to one of the agents, so for $i \in \{1, 2\}$, $V_i(X_1) + V_i(X_2) = 1$, and envy-freeness requires that $V_i(X_i) \geq 1/2$. □

We are now ready to prove Theorem 4.2.3.

Proof of Theorem 4.2.3. Consider each of the cases specified by Algorithm 2.

Case 1: $V_1(Y_{1\geq 2}) \geq 1/2, V_2(Y_{2\geq 1}) \geq 1/2$. Algorithm 2 allocates $Y_{1>2}$ to agent 1 and $Y_{2>1}$ to agent 2. The allocation made by Algorithm 2 is always efficient, since the agent who strictly prefers an interval always receives it. What is left to be shown is that the allocation is EF.

Case 1(a): $V_1(Y_{1>2}) \geq 1/2$. Algorithm 2 gives $Y_{1=2}$ to agent 2. $V_2(Y_{2\geq 1}) \geq 1/2$ by assumption. Both agents have value at least 1/2, and by Lemma 4.2.4 are not envious.
Case 1(b): $V_1(Y_{1>2}) < 1/2$. Algorithm 2 splits $Y_{1=2}$ so that agent 1 receives value exactly 1/2 after adding in $Y_{1>2}$. This must be possible since $V_1(Y_{1\geq 2}) \geq 1/2$. Agent 1 is not envious by Lemma 4.2.4. Let X_2 be the piece given to agent 2 (the remaining portion of $Y_{1=2}$ along with $Y_{2>1}$). Algorithm 2 allocates the entire interval, so by additivity, $V_1(X_2) = 1/2$. However, the piece X_2 consists only of intervals where $v_2(x) \geq v_1(x)$, so $V_2(X_2) \geq V_1(X_2) = 1/2$.

Case 2: $V_1(Y_{1\geq 2}) < 1/2$. First, note that Algorithm 2 finds an EF allocation X_1, X_2. Indeed, as before, agent 1 is not envious as $V_1(X_1) = 1/2$. Since agent 2 is given all the intervals not given to agent 1, $V_1(X_2) = 1/2$. The piece X_2 consists only of intervals where $v_2(x) > v_1(x)$, so $V_2(X_2) \geq V_1(X_2) = 1/2$.

Because $V_1(Y_{1\geq 2}) < 1/2$, envy-freeness requires us to sacrifice efficiency since we need to give agent 1 some intervals that are strictly preferred by agent 2. To show that X_1, X_2 is a maxsum EF allocation, let X_1', X_2' be any maxsum EF allocation. Define the following three pieces of cake:

$$A = X_1 \cap X_1' \cap Y_{2>1},$$
$$B = (X_1 \setminus X_1') \cap Y_{2>1},$$
$$C = (X_1' \setminus X_1) \cap Y_{2>1}.$$

A gives the intervals where both allocations lose efficiency due to giving pieces preferred by agent 2 to agent 1. B gives the intervals where X_1, X_2 loses efficiency, and C gives the intervals where X_1', X_2' loses efficiency.

Let $V_1(Y_{1\geq 2}) = 1/2 - \epsilon, \epsilon > 0$. Note that $A \cap B = \emptyset, A \cap C = \emptyset$, and $A \cup B = X_1 \cap Y_{2>1}, A \cup C = X_1' \cap Y_{2>1}$. Algorithm 2 gives agent 1 exactly value 1/2 yielding:[5]

$$V_1(A) + V_1(B) = \int_A v_1(x)dx + \int_B v_1(x)dx = \epsilon. \tag{4.8}$$

Similarly, Lemma 4.2.6 says that since X_1', X_2' is a maxsum EF allocation, agent 1 must receive value at least ϵ from its allocation of $Y_{2>1}$:

$$V_1(A) + V_1(C) = \int_A v_1(x)dx + \int_C v_1(x)dx \geq \epsilon. \tag{4.9}$$

Combining (4.8) and (4.9) yields

$$\int_C v_1(x)dx - \int_B v_1(x)dx \geq 0. \tag{4.10}$$

Let $\ell(X_1, X_2)$ denote the difference between the efficiency of a maxsum allocation (not necessarily EF) and the efficiency of X_1, X_2.

$$\ell(X_1, X_2) = \int_A (v_2(x) - v_1(x))dx + \int_B (v_2(x) - v_1(x))dx$$
$$\ell(X_1', X_2') \geq \int_A (v_2(x) - v_1(x))dx + \int_C (v_2(x) - v_1(x))dx$$

The loss for X_1', X_2' is an inequality because while Algorithm 2 gives all of $Y_{1>2}$ to agent 1, X_1', X_2' need not and may lose efficiency from those intervals as well.

[5] We slightly abuse notation and take the integral over A, B, C to signify the sum of integrals over inclusion-maximal subintervals of A, B, C respectively.

To complete the proof, recall how Algorithm 2 constructs X_1. Let r^* be the value computed in Step 2 of Algorithm 2. By definition of Algorithm 2, $X_1 \cap Y_{2>1}$ consists of all points with $R_1(x) > r^*$ and some or all of the points with $R_1(x) = r^*$. Therefore, if $x \in B$ then $R_1(x) \geq r^*$, and if $x \in C$ then $R_1(x) \leq r^*$. We conclude that

$$\begin{aligned}
&\ell(X_1', X_2') - \ell(X_1, X_2) \\
&\geq \int_C (v_2(x) - v_1(x))dx - \int_B (v_2(x) - v_1(x))dx \\
&= \int_C \left(\frac{v_1(x)}{R_1(x)} - v_1(x)\right) dx - \int_B \left(\frac{v_1(x)}{R_1(x)} - v_1(x)\right) dx \\
&\geq \left(\frac{1}{r^*} - 1\right) \left(\int_C v_1(x)dx - \int_B v_1(x)dx\right) \\
&\geq 0,
\end{aligned}$$

where the last inequality follows from (4.10). □

Interestingly, Algorithm 2 does not make specific use of the assumption that valuations are piecewise linear. In theory, it can be applied to more general classes of valuation functions, provided that the sets $Y_{1>2}, Y_{1=2}, Y_{2>1}, Y_{\geq r}$ correspond to legal pieces of cake. However, we do use the piecewise linear assumption in the next subsection.

Implementing Algorithm 2

To discuss implementation details, we need to be careful about how we represent the input to our algorithm. As discussed in Section 3.5.1, we assume that the input to our algorithm consists of the endpoints of agents' desired intervals and the slopes and intercepts of their density functions on each of these intervals. We assume that the endpoints, slopes, and intercepts can be specified with k-bit rationals. Since slopes and intercepts can be negative, in this section we take k-bit rationals to include numbers of the form $-a/b$ where a, b are k-bit rationals.

While it is tempting to apply Algorithm 2 to produce a maxsum EF allocation, there is a barrier to this approach. Even when the inputs are k-bit rational numbers, the r^* defined in Step 2 of Algorithm 2 and the boundaries of the resulting allocation may be irrational. In fact, this limitation is not specific to Algorithm 2. There are cases where the allocation computed by Algorithm 2 is the *unique* maxsum EF allocation and has irrational boundaries.

Theorem 4.2.7. *There exist piecewise linear valuations whose interval boundaries, slopes,*

and intercepts are all rational numbers yet whose maxsum EF allocations can only be specified with irrational numbers.

Proof. Suppose agent 2 has value density function $v_2(x) = 2x$ over the entire interval $[0, 1]$ and agent 1 has value density function

$$v_1(x) = \begin{cases} \frac{1}{2} & \text{if } 0 \leq x \leq \frac{1}{4} \\ \frac{32x+1}{18} & \text{if } \frac{1}{4} < x \leq 1 \end{cases} \tag{4.11}$$

The first step of Algorithm 2 will give $[0, 1/4]$ to agent 1 and $[1/4, 1]$ to agent 2 since agent 1 has higher density on $[0, 1/4]$ and agent 2 has higher density on $[1/4, 1]$. At this point, agent 1 will be envious of agent 2, having only value $(1/2)(1/4) = 1/8$ for its piece (and therefore value $7/8$ for agent 2's piece).

The next step of Algorithm 2 takes parts of $[1/4, 1]$ from agent 2 and gives them to agent 1 until agent 1 obtains value exactly $1/2$. In particular, the algorithm allocates some interval of the form $[1/4, a^*]$ to agent 1 since $R_1(x)$ is decreasing on $[1/4, 1]$. For agent 1 to obtain value exactly $1/2$, the piece $[1/4, a^*]$ must be worth exactly $3/8$ to agent 1 (since $[0, 1/4]$ is worth exactly $1/8$). Thus, a^* will satisfy

$$\int_{\frac{1}{4}}^{a^*} v_1(x)dx = \frac{3}{8}.$$

Solving this equation, we find that a^* is given by

$$a^* = \frac{-1 + 3\sqrt{57}}{32}.$$

Therefore, the maxsum EF allocation computed by Algorithm 2 has irrational boundaries and we cannot hope to specify it with a finite number of bits. To finish the proof, note that in this example, there are no ties or places where we can specify multiple different allocations as $R_1(x)$ is strictly increasing on $[1/4, 1]$. We can therefore argue that the allocation produced by Algorithm 2 is the unique maxsum EF allocation since any other allocation would result in transfers of intervals with lower $R_1(x)$ from agent 2 to agent 1, leading to more efficiency loss (using similar arguments as the proof of Theorem 4.2.3. As a result, we cannot get around the problem of irrational boundaries by selecting a particular maxsum EF allocation. □

As a result, it is necessary to resort to approximation. Indeed, we relax envy-freeness

by considering approximately EF allocations. Specifically, an allocation is *ϵ-EF* if for all $i, j \in N$, $V_i(X_i) \geq V_i(X_j) - \epsilon$ (see, e.g., Lipton et al. [2004]). The following theorem formally presents our approximation guarantees.

Theorem 4.2.8. *Assume that there are two agents with piecewise linear valuations. For any $\epsilon > 0$ there is an algorithm that runs in time polynomial in the input and $\log(1/\epsilon)$, and finds an ϵ-EF allocation A' such that $e(A') \geq e(A)$, where A is a maxsum EF allocation.*

Proof (Sketch). The full proof is deferred to the appendix of this chapter, but we give a brief sketch here. In the case considered in Step 1 of Algorithm 2, we would like to find a point x^* such that

$$V_1(([0, x^*] \cap Y_{1=2}) \cup Y_{1>2}) = 1/2.$$

Using binary search over $[0, 1]$, we find a point x that is smaller but very close to x^*. It is then possible to bound the envy, while the resulting allocation is at least as efficient as any maxsum EF allocation. In Step 2 of Algorithm 2, we need to search for a ratio r close to r^*. This is more subtle, because very small differences in $|r - r^*|$ can lead to significant differences in the derived value when there is a long interval with constant value ratio. Fortunately, in this problematic case it can be shown that r^* is a rational, and hence it is sufficient to find the rational r closest to r^*. This can be done using a delicate search over rationals, via techniques due to Papadimitriou [1979]. □

4.2.4 General Valuations

In this section we give a method for handling general valuation functions (under some mild conditions) and for any number of agents. We approximate general valuation functions with piecewise constant valuations and leverage Algorithm 1. We construct an allocation that is ϵ-EF and whose efficiency is within ϵ of any maxsum EF allocation.

Lemma 4.2.9. *Given $\epsilon > 0$ and value density functions $v_1, \ldots, v_n$, suppose that $v'_1, \ldots, v'_n$ are piecewise constant value density functions such that for all $i \in N$,*

$$v_i(x) \leq v'_i(x) \leq v_i(x) + \epsilon/2. \tag{4.12}$$

Let $A = (X_1, \ldots, X_n)$ be a maxsum EF allocation with respect to valuations V_i (induced by v_i), and let $A' = (X'_1, \ldots, X'_n)$ be a maxsum $\epsilon/2$-EF allocation with respect to valuations V'_i (induced by v'_i). Then A' is ϵ-EF and $e(A') \geq e(A) - \epsilon/2$.

Proof. To show that A' is ϵ-EF with respect to $V_1, \ldots, V_n$, let $i, j \in N$, and note that A' is $\epsilon/2$-EF with respect to V_i', so $V_i'(X_i') \geq V_i'(X_j') - \epsilon/2$. Thus, using (4.12), $V_i(X_i') \geq V_i'(X_i') - \epsilon/2 \geq V_i'(X_j') - \epsilon \geq V_i(X_j') - \epsilon$.

For the second part of the lemma, we first claim that

$$\sum_{i=1}^{n} V_i'(X_i') \geq \sum_{i=1}^{n} V_i'(X_i). \tag{4.13}$$

To prove this, it is sufficient to show that A is $\epsilon/2$-EF with respect to $V_1', \ldots, V_n'$, as A' represents the maxsum $\epsilon/2$-EF allocation with respect to $V_1', \ldots, V_n'$ (so A cannot possibly provide more welfare). Indeed, A is EF with respect to $V_1, \ldots, V_n$, and hence

$$V_i'(X_i) \geq V_i(X_i) \geq V_i(X_j) \geq V_i'(X_j) - \epsilon/2.$$

Next, it holds that

$$\begin{aligned}
\sum_{i=1}^{n} V_i(X_i') &= \sum_{i=1}^{n} \int_{X_i'} v_i(x)dx \\
&\geq \sum_{i=1}^{n} \int_{X_i'} (v_i'(x) - \epsilon/2)dx \\
&= \left(\sum_{i=1}^{n} \int_{X_i'} v_i'(x)dx\right) - \epsilon/2 \\
&= \left(\sum_{i=1}^{n} V_i'(X_i')\right) - \epsilon/2.
\end{aligned} \tag{4.14}$$

Now, the assertion that $e(A') = \sum_{i=1}^{n} V_i(X_i') \geq (\sum_{i=1}^{n} V_i(X_i)) - \epsilon/2 = e(A) - \epsilon/2$ directly follows by combining Equations (4.13), (4.14), and (4.12). □

Given piecewise constant value density functions $v_1', \ldots, v_n'$ that satisfy (4.12), it is easy to find a maxsum $\epsilon/2$-EF allocation A' by applying Algorithm 1 to these valuations, where the envy-freeness constraint (4.3) is relaxed by $\epsilon/2$.

To find $v_1', \ldots, v_n'$ that satisfy (4.12), we assume that $v_1, \ldots, v_n$ are K-Lipschitz, i.e., for all $x, y \in [0, 1]$,

$$|v_i(x) - v_i(y)| \leq K \cdot |x - y|.$$

Now, split $[0, 1]$ into $\lceil 4K/\epsilon \rceil$ intervals of size at most $\epsilon/(4K)$. Let $S = \{k/2^p : k \in [0, M2^p]\}$, where M is an upper bound on $v_i(x)$ for all $i \in N$ and $x \in [0, 1]$, and p will be specified later.

For each interval I and agent i, let $v^*(I) = \max_{x \in I} v_i(x)$. For all $x \in I$ let $v_i'(x) = s^*(I)$, where $s^*(I) = \min\{s \in S : s \geq v^*(I)\}$.

The K-Lipschitz condition ensures that the density function varies by at most $\epsilon/4$ on each interval. If we take $p = \lceil 2 + \log(1/\epsilon) \rceil$, then $s^*(I) - v^*(I) \leq \epsilon/4$, and v_i' satisfies condition 4.12.

While the K-Lipschitz condition rules out valuation density functions with discontinuities, our results extend to valuation density functions with a finite number of discontinuities that are K-Lipschitz on each continuous subinterval. In particular, we can use the described procedure separately on each continuous subinterval to find v_i' that satisfy (4.12). We have the following theorem.

Theorem 4.2.10. *Assume that there are n agents with value density functions $v_1, \ldots, v_n$ that have a finite number of discontinuities, are K-Lipschitz on each continuous subinterval, and have maximum value M. For any $\epsilon > 0$, there is an algorithm that runs in time polynomial in $n, \log M, K, 1/\epsilon$ and computes an ϵ-EF allocation whose efficiency is within ϵ of any maxsum EF allocation.*

4.2.5 Discussion

Normalization

Though we assume normalization for ease of exposition, the results in this section extend naturally to unnormalized valuations. With unnormalized valuations, we let each agent have a different value for the entire cake $[0, 1]$. For piecewise constant valuations (Section 4.2.2), the LP still returns the maxsum EF allocation even if agent valuations are not normalized. For piecewise linear valuations and two agents (Section 4.2.3), the intuition of trying to swap intervals with high ratios before intervals with smaller ratios still applies when agent valuations are not normalized. The difference is that the cases need to be separated based on whether agent values are at least $V_i([0, 1])/2$ instead of $1/2$. For approximating general valuations, the same techniques apply when agent valuations are not normalized. We first approximate the valuations with piecewise constant valuations. Then we find a maxsum $\epsilon/2$-EF allocation using the LP for piecewise constant valuations. Then we argue that this allocation is approximately EF with respect to the true valuations and approximately maximizes social welfare.

Comparison of Section 4.2.3 with Theorem 4.2.10

Given the rather strong Theorem 4.2.10, one may wonder in what way the results of Section 4.2.3 are superior. In fact, for the (interesting, we believe) case of two agents with piecewise linear valuations, the method of Section 4.2.3 has two technical advantages. First, it produces an ϵ-EF allocation that is as efficient as the maxsum EF allocation. Second, Theorem 4.2.8 provides running time that is polynomial in the representation and therefore logarithmic in the slope of the valuation functions, as the slope is specified by $O(k)$-bit rationals. In contrast, the running time in Theorem 4.2.10 is polynomial in the slope (since the maximum slope determines the Lipschitz constant), and hence exponential in the representation. Finally, note that piecewise linear (rather than constant) valuation functions can in theory be used to approximate *general* valuations, making it possible to relax the assumptions of Theorem 4.2.10 (when there are only two agents).

Proportionality

Envy-freeness can be replaced with the weaker notion of proportionality in all of our results. The purpose of our focus on envy-freeness is to simplify the exposition. As mentioned in Section 3.2, any EF allocation is proportional, and for the case of two agents the two notions coincide. Using the last observation, the results of Section 4.2.3 immediately hold for proportionality. The results of Section 4.2.2 can easily be adapted by modifying (4.3). This can then be used to obtain results similar to Section 4.2.4.

Direct Revelation

We have assumed a direct revelation model of cake cutting throughout this section. Of course, in Section 4.2.4, which deals with general valuations, we cannot adopt this model. However, notice that Theorem 4.2.10 merely requires finding values that are close to $v_i(x)$ for a polynomial number of points $x \in [0, 1]$; an implicit, reasonable assumption is that the valuation information at these points can be elicited from agents.

4.3 Properties of Maxsum Fair Allocations

Intuitively, a maxsum fair allocation is superior to an arbitrary fair allocation, for any fairness criterion. Nevertheless, we do not know *how good* maxsum fair allocations are; can one argue that they are truly more desirable than other allocations? Moreover, there are

several notions of fairness to choose from; under which notion should one optimize social welfare?

In this section, we consider different maxsum fair allocations. In particular, we focus on maxsum EF, EQ and (EF and EQ) allocations. Specifically, we ask whether these allocations are Pareto-efficient (PE) among the set of all possible allocations. We then consider the question of whether we can say anything of the social welfare of these different allocations. We do not consider maxsum proportional allocations, as they are not interesting from the perspective of Pareto-efficiency. Indeed, it is easy to see that any maxsum proportional allocation must be Pareto-efficient among all allocations. If not, we could make all agents weakly better off, increasing social welfare while preserving proportionality.

We first observe that, if there are only two agents, PE is guaranteed for maxsum EF allocations, maxsum EQ allocations, and even maxsum EF and EQ allocations (i.e., allocations that are maxsum among allocations that are both EF and EQ).

Our other results are more subtle and hinge on the structure of agents' valuation functions. We consider the special case of piecewise uniform valuations and piecewise constant valuations. We show that under piecewise uniform valuations, maxsum EF allocations are always PE whereas there are cases where all maxsum EQ and maxsum EF+EQ allocations are not PE. Under piecewise constant valuations, there are examples with three agents such that all maxsum EF allocations are also not PE.

We then move onto comparing the social welfare under maxsum EF and maxsum EQ allocations. We show that under piecewise linear valuations the social welfare of a maxsum EF allocation is at least as great as the social welfare of a maxsum EQ allocation. We also extend this result to general valuation functions albeit only approximately, in that (i) we optimize among allocations that are EF up to ϵ, and (ii) the inequality holds up to ϵ.

4.3.1 Pareto Efficiency of Maxsum Allocations

In this section, we study the Pareto efficiency of maxsum allocations. In particular, we establish the Pareto efficiency of maxsum EF, EQ, and EF+EQ allocations in the case of two agents and general valuations, and complement this result by showing that for three agents or more, these allocations are not necessarily Pareto efficient.

Two Agents, General Valuations

The two-agent case has special significance (for example, in the context of divorce settlements), so we give special consideration to this case.

Theorem 4.3.1. *For general valuations and two agents, every maxsum EF, EQ, or EF+EQ allocation is PE.*

Before proving Theorem 4.3.1, we introduce the notion of ratio-based allocations for the two-agent setting. Ratio-based allocations generalize the type of allocation produced by Algorithm 2 in Section 4.2.3. We adopt the same notation for R_i, $Y_{i \ op \ j}$, and $Y_{R_i \ op \ r}$ introduced in that section. In addition, we let Y_1 and Y_2 denote the intervals on which only agent 1's density is positive and only agent 2's density is positive respectively.

Definition 4.3.2. An allocation $A = (A_1, A_2)$ is *ratio-based* if $Y_1 \subseteq A_1, Y_2 \subseteq A_2$ and either one of the following holds:

- There exists an $r^* \in [0, 1]$ such that

$$A_1 = Y_{1>2} \cup Y_{R_1 > r^*} \cup C,$$

  where $C \subseteq Y_{R_1 = r^*}$.

- There exists an $r^* \in [0, 1]$ such that

$$A_2 = Y_{2>1} \cup Y_{R_2 > r^*} \cup C,$$

  where $C \subseteq Y_{R_2 = r^*}$.

We refer to agent 1 as the *receiving agent* in the first case and agent 2 as the receiving agent in the second case. We refer to r^* as the *critical ratio*.

In a ratio-based allocation, the receiving agent is always allocated intervals that it strictly desires, as well as some intervals weakly desired by the other agent. For the special case where the critical ratio is 1, both agents can be seen as receiving agents. In this case, the allocation is efficient since all intervals are allocated to agents who weakly prefer the interval. When the critical ratio is less than 1, there is a unique receiving agent i that receives all intervals it weakly desires ($Y_{i \geq 3-i}$) along with some intervals strictly desired by the other agent. This necessarily results in a loss of welfare relative to the efficient allocation. However, ratio-based allocations minimize the obtained loss. This is formalized in the following lemma.

Lemma 4.3.3. *Let $A = (A_1, A_2)$ be a ratio-based allocation with agent 1 as the receiving agent such that $v = V_1(A_1) \geq V_1(Y_{1 \geq 2})$. It holds that:*

1. *For every allocation* $A' = (A'_1, A'_2)$ *such that* $V_1(A'_1) = v$, $sw(A) \geq sw(A')$.

2. *For every allocation* $A' = (A'_1, A'_2)$ *such that* $V_1(A'_1) > v$, $sw(A) > sw(A')$.

An analogous assertion holds for agent 2.

Proof (sketch). The proof of the lemma closely resembles the proof of Theorem 4.2.3. Among all allocations that grant agent 1 value v, the allocation that maximizes welfare is one in which agent 1 is first allocated all the intervals it strictly desires, and then, possibly, intervals that are strictly desired by agent 2, in a decreasing order of $R_i(x)$. The first part entails no loss in welfare. The second part may entail some loss, but allocating these intervals in a decreasing order of $R_i(x)$ ensures that this is the lowest possible loss. In addition, if agent 1 receives value greater than v, it must come from additional intervals that are strictly desired by agent 2. This entails a greater loss in welfare. □

The following is an immediate corollary of Lemma 4.3.3.

Lemma 4.3.4. *Every ratio-based allocation is PE.*

We are now ready to prove Theorem 4.3.1.

Proof of Theorem 4.3.1. We address the three different allocation types.

Maxsum EF. We distinguish between two cases, as follows. If there exists an EF allocation that is efficient, then every maxsum EF allocation is trivially PE. Otherwise, there must exist an agent i such that $V_i(Y_{i \geq 3-i}) < 1/2$ (Step 2 of Algorithm 2). While Algorithm 2 focuses on piecewise linear valuation functions, the algorithm works for general valuation functions. Wlog, suppose $V_1(Y_{1 \geq 2}) < 1/2$. Theorem 4.2.3 establishes the existence of a ratio-based allocation that gives agent 1 value of exactly 1/2 and is maxsum EF. Let $A = (A_1, A_2)$ be such an allocation. By Lemma 4.3.4, A is PE. Let $A' = (A'_1, A'_2)$ be another maxsum EF allocation. In what follows we show that A' is PE. We distinguish between three cases.

1. If $V_1(A'_1) = 1/2$, then, since A' is maxsum EF, it follows that $V_2(A'_2) = V_2(A_2)$. In this case, the fact that A is PE implies that A' is PE as well.

2. If $V_1(A'_1) < 1/2$, then $V_1(A'_2) < 1/2$ (otherwise, contradicting EF). It follows by Lemma 4.2.5 that A' is not maxsum EF, a contradiction.

3. If $V_1(A'_1) > 1/2$, then we get $V_1(A'_1) > 1/2 > V_1(Y_{1 \geq 2})$. It follows by Lemma 4.3.3 that $sw(A') < sw(A)$, in contradiction to A' being a maxsum EF allocation.

Maxsum EQ. We distinguish between two cases.

- $V_1(Y_{1\geq 2}) \geq V_2(Y_{2>1})$ and $V_2(Y_{2\geq 1}) \geq V_1(V_{1>2})$. In this case we show that there exists a maxsum EQ allocation that is efficient. This, in turn, implies that every maxsum EQ allocation is PE. In particular, allocate $Y_{1>2}$ to agent 1, $Y_{2>1}$ to agent 2, and split $Y_{1=2}$ such that the agents' values for their pieces are equal. To see why this is feasible, note that if we give $Y_{1=2}$ to agent 1 in its entirety, then agent 1 has a greater value. On the other hand, if we give all of $Y_{1=2}$ to agent 2, then agent 2 has a greater value. Therefore, there must exist some allocation of $Y_{1=2}$ that equalizes their values. This allocation is EQ and efficient.

- Wlog, suppose $V_1(Y_{1\geq 2}) < V_2(Y_{2>1})$. We claim that in this case there exists a ratio-based allocation with agent 1 as the receiving agent that is EQ. To see this, note that as the critical ratio decreases from 1 to 0, agent 1 goes from receiving all of $Y_{1\geq 2}$ to receiving the entire cake, i.e., from a value of $V_1(Y_{1\geq 2})$ to a value of 1. On the other hand, agent 2 goes from receiving all of $Y_{2>1}$ to receiving none of the cake, i.e., from value $V_2(Y_{2>1}) > V_1(Y_{1\geq 2})$ to 0. Therefore, the agents' values must cross at some point, and the assertion follows. By Lemma 4.3.4 this allocation is PE, and hence maxsum EQ. Clearly, any maxsum EQ allocation must grant each agent the same value as in the ratio-based maxsum EQ allocation. It follows that every maxsum EQ allocation is PE.

Maxsum EF+EQ. In every maxsum EQ allocation, both agents receive value at least 1/2. To see why this is true, consider the allocation given by cut and choose. Each agent obtains value at least 1/2 in this allocation, and we can make it EQ by destroying intervals given to the agent with higher value. Since both agents receive value at least 1/2, the maxsum EQ allocation is also EF. It follows that for two agents, the set of maxsum EF+EQ allocations coincides with the set of maxsum EQ allocations, for which the assertion of the theorem is proved above. □

Any Number of Agents, Restricted Valuations

We next turn to investigate maxsum EF, maxsum EQ, and maxsum EF+EQ allocations under restricted valuations, but for any number of agents. As it turns out, at least under piecewise uniform valuation functions, maxsum EF allocations are always PE whereas maxsum EQ and maxsum EF+EQ allocations may not be.

Theorem 4.3.5. *For piecewise uniform valuations, every maxsum EF allocation is PE.*

Proof sketch. When agent valuations are piecewise uniform, a sufficient condition for PE is that all intervals desired by at least one agent are allocated to an agent that has positive density on the entire interval. To see why this is true, recall that when agents have piecewise uniform valuations, their total value is exactly determined by the total length of desired intervals they receive. If all desired intervals are allocated to agents with positive density, then an allocation that makes everyone weakly better off and one agent strictly better off cannot exist because this would require additional desired lengths to be created. It remains to show that a maxsum EF allocation must have this property.

Suppose that a maxsum EF allocation $A = (A_1, \ldots, A_n)$ allocates some intervals to agents that do not desire them or discards intervals altogether. Let X' denote these intervals. Under piecewise uniform valuations, we can split X' into subintervals on which agent densities are constant, and then give each agent a $1/n$ share of each of these subintervals. We can append this allocation of X' to A. Envy is not created, because each agent i has value exactly $(1/n)V_i(X')$ for every piece in this allocation, but social welfare increases, contradicting the assumption that A is maxsum. □

Theorem 4.3.6. *For piecewise uniform valuations and three agents, there are valuation functions where all maxsum EQ and EF+EQ allocations are not PE.*

Proof. Consider the following valuations. Agents 1 and 2 desire $[0, 0.1]$ and agent 3 desires all of $[0, 1]$. A maxsum EQ or maxsum EF+EQ allocation must split $[0, 0.1]$ between agents 1 and 2 and allocate $[0, 1]$ to agent 3 so that agent 3 receives value exactly 0.5. This is not PE because we can split $[0, 0.1]$ between agents 1 and 2 and give agent 3 all of $[0.1, 1]$. □

While there are cases where no maxsum EQ or EF+EQ allocation is PE under piecewise uniform valuations, we need to move to piecewise constant valuations in order to find cases where no maxsum EF allocation is PE.

Theorem 4.3.7. *For piecewise constant valuations and three agents, there are cases where no maxsum EF allocation is PE.*

Proof (sketch). Consider the following valuations for three agents. The cake is split into three equal intervals, i.e., $[0, 1/3], [1/3, 2/3], [2/3, 1]$. Each agent's value densities are constant on each of these intervals. Agent 1 values interval 1 at 51/101 and interval 2 at 50/101, i.e., has densities 153/101 on interval 1 and 150/101 on interval 2. Agent 2 values interval 1 at 50/101 and interval 2 at 51/101. Agent 3 values interval 1 at 51/111, interval 2 at 10/111, and interval 3 at 50/111 (Figure 4.2).

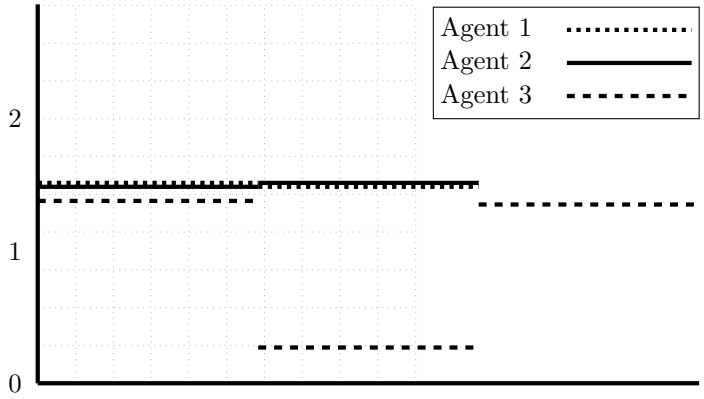


Figure 4.2: Value density functions for example where maxsum EF is not PO.

The efficient solution gives interval i to agent i. However, this allocation is not EF because agent 3 envies agent 1. It can be shown via the LP formulation for the computation of a maxsum EF allocation that to reduce this envy, there is a maxsum EF allocation that gives agent 1 a share of intervals 1 and 2, agent 2 a share of intervals 1 and 2, and agent 3 all of interval 3. Further, an examination of this LP shows that this allocation is the unique maxsum EF allocation. However, this allocation is not PE because agent 1 receives shares of interval 2 (where agent 2 has higher density) while agent 2 receives shares of interval 1 (where agent 1 has higher density). As a result, there is a way to swap agent 1's shares of interval 2 with agent 2's shares of interval 1 that leaves both agents better off. The full proof is given in the appendix of this chapter. □

4.3.2 Maxsum EQ vs. Maxsum EF Allocations

In this section, we show that for piecewise linear valuations, a maxsum EF allocation has social welfare at least as large as any maxsum EQ allocation. We obtain an approximate version of this result for general valuation functions.

Denote the social welfare of a maxsum EF (resp., EQ) allocation by OPT_{EF} (resp., OPT_{EQ}). Note that the two-agent version of the inequality $\text{OPT}_{\text{EQ}} \leq \text{OPT}_{\text{EF}}$, for any valuation functions, follows from the fact that a maxsum EQ allocation is also EF, which was established in passing in the proof of Theorem 4.3.1. As a recap, both agents receive value at least 1/2 in a maxsum EQ allocation, and for two agents, this is a sufficient condition for envy-freeness.

For three agents, this argument no longer holds, even in the case of piecewise constant valuations: a maxsum EQ allocation must give utility at least 1/3 to each agent, but this

does not imply EF. For example, consider the piecewise uniform valuations where agents 1 and 2 value the whole cake (with density 1) and agent 3 only values $[0.8, 1]$ (with density 5). A maxsum EQ allocation would be to give agent 1 $[0, 5/11]$, agent 2 $[5/11, 10/11]$, and agent 3 $[10/11, 1]$. Each agent receives value $5/11$, yet agent 3 envies agent 2.

Another interesting (but common) feature of this example is that $\text{OPT}_{\text{EQ}} < \text{OPT}_{\text{EF}}$, with a strict inequality. One EF allocation is to give $[0.8, 1]$ to agent 3 and split $[0, 0.8]$ between agents 1 and 2. This has social welfare of 1.8 compared to the maxsum EQ welfare of $15/11 \approx 1.364$.

Having built some intuition, we next present the main result of this section. An ϵ-EF allocation is one where $V_i(A_i) \geq V_i(A_j) - \epsilon$ for all $i, j \in N$. Let $\text{OPT}_{\epsilon\text{-EF}}$ denote the social welfare under a maxsum ϵ-EF allocation.

Theorem 4.3.8. *For piecewise linear valuations,*

$$OPT_{EQ} \leq OPT_{EF}.$$

Moreover, for general valuation functions and any $\epsilon > 0$,

$$OPT_{EQ} \leq OPT_{\epsilon\text{-}EF} + \epsilon.$$

The proof of Theorem 4.3.8 relies on a connection between piecewise linear valuation functions and market equilibria for a collection of divisible goods inspired by Reijnierse and Potters [1998]. Before we begin the proof, we draw this connection and cite the relevant results from the market equilibria literature required in the proof.

A *linear Fisher market* is a market where agents $N = \{1, \ldots, n\}$ have additive, linear utility functions for a set $G = \{1, \ldots, m\}$ of divisible goods. Each agent $i \in N$ is given a budget e_i and has a utility u_{ij} for each good $j \in G$. A *feasible allocation* gives a fraction x_{ij} of good j to agent i such that no good is over-allocated. The agent's total utility from an allocation x_{ij} is $\sum_j u_{ij}x_{ij}$. When agent valuations are piecewise linear, utilities in a feasible Fisher market allocation can be replicated in the cake cutting setting.

Lemma 4.3.9. *Let $A_1, \ldots, A_n$ be an allocation in the cake cutting setting. Define a Fisher market with the same agents, and a good j corresponding to each A_j and $u_{ij} = V_i(A_j)$. Let x_{ij} be a feasible allocation of goods in the Fisher market. There exists an allocation $A'_1, \ldots, A'_n$ such that $V_i(A'_j) = u_{ij}x_{ij}$. In other words, we can replicate agent utilities in the*

Fisher market with an allocation of the cake.

Proof. Given a feasible allocation x_{ij} in the Fisher market, create an allocation $A'_1, \ldots, A'_n$ as follows. For each original piece A_j, split A_j into subintervals on which every agent's value density function is linear (this is possible since agent value densities are piecewise linear).

We would like to give each agent i a piece of A_j that it values at $x_{ij}V_i(A_j)$. Since the valuations are linear, this can be achieved by giving agent i two equally-sized pieces from each linear subinterval. For a given linear subinterval, process each agent one by one. For each next agent, give the agent a $\frac{x_{ij}}{2}$ fraction of the remaining interval starting from the left and moving right, and a $\frac{x_{ij}}{2}$ fraction of the remaining interval starting from the right and moving left. Since the linear utilities are symmetric, agent i's value from its share of A_j is $x_{ij}V_i(A_j) = x_{ij}u_{ij}$, and summing over all intervals establishes the assertion of the lemma. □

Linear Fisher markets have the following very special properties (see e.g., Vazirani [2007]).

Theorem 4.3.10. *Consider a linear Fisher market where agent i has budget e_i, $\sum_{i\in N} e_i = 1$, and each good gives at least one agent positive utility. There exists a price vector $p = (p_1, \ldots, p_{|G|})$, $p_j > 0$, $\sum_{j\in G} p_j = 1$, and a feasible allocation x_{ij} such that:*

1. $\forall j \in G,\ \sum_{i\in N} x_{ij} = 1$,
2. $\forall i \in N, j \in G$, *If* $x_{ij} > 0$, *then* $j \in \operatorname{argmax}_{j'}(u_{ij'}/p_{j'})$,
3. $\forall i \in N,\ \sum_{j\in G} p_j x_{ij} = e_i$.

Leveraging this result, we prove Theorem 4.3.8.

Proof of Theorem 4.3.8. Begin with a maxsum EQ allocation $A^* = (A^*_1, \ldots, A^*_n)$. Construct a Fisher market where good j corresponds to A^*_j, $u_{ij} = V_i(A^*_j)$ and each agent has budget $e_i = 1/n$. Let p, x_{ij} be the price vector and feasible allocation guaranteed by Theorem 4.3.10. Consider the allocation $A'_1, \ldots, A'_n$ described in Lemma 4.3.9. We need to show that this allocation is EF and yields total welfare weakly greater than that of the original maxsum EQ allocation. Due to Lemma 4.3.9, we can relate the values for $A'_1, \ldots, A'_n$ to the utilities in the Fisher market.

The proof that $A'_1, \ldots, A'_n$ is EF appears in Reijnierse and Potters [1998]; the next equations replicate it for completeness. Let $u_i^* = \max_k(u_{ik}/p_k)$.

$$\begin{aligned} V_i(A'_i) &= \sum_k u_{ik}x_{ik} = \sum_k \frac{u_{ik}}{p_k}p_k x_{ik} \\ &= \sum_k u_i^* p_k x_{ik} = u_i^*/n \\ V_i(A'_j) &= \sum_k u_{ik}x_{jk} = \sum_k \frac{u_{ik}}{p_k}p_k x_{jk} \\ &\leq \sum_k u_i^* p_k x_{jk} = u_i^*/n \end{aligned}$$

It remains to show that $\sum_i V_i(A'_i) \geq \sum_i V_i(A_i^*)$. Suppose $V_i(A_i^*) = C$ for all $i \in N$; then $\text{OPT}_{\text{EQ}} = \sum_i V_i(A_i^*) = nC$. u_i^* maximizes u_{ik}/p_k, so u_i^* is at least u_{ii}/p_i, the utility to price ratio for the good in the Fisher market corresponding to A_i^*. Therefore, $V_i(A'_i) = u_i^*/n \geq u_{ii}/(np_i) = C/(np_i)$.

Then,

$$\text{OPT}_{\text{EF}} \geq \sum_i V_i(A'_i) \geq \sum_i \frac{C}{np_i} = \frac{C}{n}\sum_i \frac{1}{p_i}.$$

Since $\sum_i p_i = 1$, $\sum_i(1/p_i)$ is minimized by $p_i = 1/n$ for each i and is at least n^2. Therefore,

$$\text{OPT}_{\text{EF}} \geq \frac{C}{n}\sum_i \frac{1}{p_i} \geq \frac{C}{n}n^2 = nC = \text{OPT}_{\text{EQ}}.$$

Next we establish our result for general valuation functions $V_1, \ldots, V_n$ (with Riemann integrable value density functions). For $\epsilon > 0$, Riemann integrability of $v_1, \ldots, v_n$ implies that for all $i \in N$ there are $0 = x_1 < \cdots < x_m = 1$ such that the upper Darboux sum of v_i satisfies

$$\begin{aligned} 1 &= \int_{x=0}^{1} v_i(x)dx \\ &\leq \sum_{k=1}^{m}\left[(x_k - x_{k-1}) \cdot \left(\sup_{x\in[x_{k-1},x_k]} v_i(x)\right)\right] \leq 1 + \frac{\epsilon}{n}. \end{aligned} \tag{4.15}$$

For every $k = 1, \ldots, m$ and every $y \in [x_{k-1}, x_k]$, let $v'_i(y) = \sup_{x\in[x_{k-1},x_k]} v_i(x)$. We claim that the corresponding piecewise constant valuation functions $V'_1, \ldots, V'_n$ approximate the

original valuation functions in the sense that for every piece of cake X,[6]

$$V_i(X) \leq V_i'(X) \leq V_i(X) + \frac{\epsilon}{n}. \tag{4.16}$$

Indeed, the left hand side of the inequality is trivial, and the right hand side follows from Equation (4.15) and the fact that $v_i'(x) \geq v_i(x)$ for all $x \in [0,1]$:

$$\begin{aligned} V_i'(X) - V_i(X) &= \int_X (v_i'(x) - v_i(x))dx \\ &\leq \int_{x=0}^{1} (v_i'(x) - v_i(x))dx \leq \frac{\epsilon}{n}. \end{aligned}$$

Assume as before that the maxsum EQ allocation A^* satisfies $V_i(A_i^*) = C$ for all $i \in N$. It therefore holds that $V_i'(A_i^*) \geq C$ for all $i \in N$. Using the same arguments as before (and the fact that piecewise constant valuations are in particular piecewise linear), there exists an allocation A' that is EF with respect to $V_1', \ldots, V_n'$ and satisfies

$$\sum_{i \in N} V_i'(A_i') \geq nC = \sum_{i \in N} V_i(A_i^*) = \mathrm{OPT}_{\mathrm{EQ}}.$$

Equation (4.16) directly implies that the allocation A' is ϵ-EF (in fact, (ϵ/n)-EF) with respect to the valuations $V_1, \ldots, V_n$. Therefore, it holds that

$$\begin{aligned} \mathrm{OPT}_{\epsilon\text{-EF}} &\geq \sum_{i \in N} V_i(A_i') \geq \sum_{i \in N} \left(V_i'(A_i') - \frac{\epsilon}{n} \right) \\ &= \sum_{i \in N} V_i'(A_i') - \sum_{i \in N} \frac{\epsilon}{n} \geq \mathrm{OPT}_{\mathrm{EQ}} - \epsilon. \end{aligned}$$

□

4.3.3 Discussion

Normalization

As in the previous section, we assume in this section that agent valuations are normalized so that $V_i([0,1]) = 1$ for all i. The PE results for two agents naturally extend to the setting where agent valuations are not normalized. In particular, ratio-based allocations still play an important role and maxsum fair allocations will be ratio-based. For settings with any number of agents, the positive PE results for piecewise uniform valuations still hold. The

[6] It may be the case that $V_i'([0,1]) > 1$.

other PE results are mostly impossibilities and clearly continue to hold if we expand the set of allowable valuations. The results relating the welfare of maxsum EF and maxsum EQ do not rely on agent valuations being normalized and should continue to hold when valuations are not normalized.

Which Allocations to Choose?

Our work can be seen as another step on the path to identifying the most desirable allocations of divisible goods. In recent work, Brams et al. [2012b] coined the term *perfect allocations* to describe allocations that are PE, EF, and EQ. Unfortunately, they show that such allocations may not exist when there are three or more agents, however many cuts are allowed. We therefore argue that maximizing social welfare under a subset of these three properties provides an especially appealing solution, but as we discuss below, there are trade-offs among the different properties.

One may wonder, in light of Theorem 4.3.8, whether a maxsum EF allocation is superior to a maxsum EQ allocation. While we believe that this is often true, we wish to add a caveat. Consider an example where there are three agents with value density functions $v_1(x) = v_2(x) = 1$, $v_3(x) = 2x$. A maxsum EF allocation gives $[0, 1/3]$ to agent 1, $[1/3, 2/3]$ to agent 2, and $[2/3, 1]$ to agent 3, for a sum of $1/3 + 1/3 + 5/9 \approx 1.22$. This allocation also happens to be PE. But there is a maxsum EQ allocation that is also EF (by dividing the left portion of the cake between agents 1 and 2 in a way that 3 does not envy either) and gives each agent a value of roughly 0.39, for a slightly lower sum of 1.17. The latter allocation seems more desirable, because it maximizes the minimum value to the agents. Indeed, the EF allocation creates significant inequity between agents 1 and 2, on the one hand, and agent 3 on the other (1/3 vs. 5/9); this 67% difference in values in exchange for only a 4% higher social welfare, compared with EQ (1.22 vs. 1.17), arguably tips the balance in favor of the maxsum EQ allocation: it not only gives all agents the same "fair share," unlike the maxsum EF allocation, but it is also EF.

4.4 Summary and Future Work

In this chapter, we explore maxsum fair allocations when agents have piecewise uniform, piecewise constant, and piecewise linear valuations. Section 4.2 examines the algorithmic problem of computing these maxsum fair allocations. The main results are an exact algorithm for piecewise constant valuations and an approximate algorithm for piecewise linear

and more general valuations.

Section 4.3 analyzes the properties of maxsum fair allocations in an attempt to shed some light on which allocations should be chosen in a cake cutting setting. We show that maxsum fair allocations are imperfect, and we crystallize some of the trade-offs among them. Our contributions inform the discussion of good methods for resource allocation by (i) ruling out the possibility that maxsum EF allocations are always superior to other allocations (by showing that they may not be PE), and (ii) demonstrating that moving from EF to the egalitarian notion of EQ can only decrease social welfare.

4.4.1 Future Work

1. We have shown that maxsum EF allocations may not be PE, and hence one may consider choosing an allocation that Pareto-dominates the maxsum EF allocation. However, in the examples that we have been able to construct where the maxsum EF allocation is indeed not PE, the difference in social welfare between the maxsum EF allocation and its Pareto-dominating allocation is very small. Bounding this difference (or ratio) remains an open question (which is somewhat related to work on the so-called *price of fairness* [Caragiannis et al., 2009]), but if it is indeed always small, we would argue that preserving EF is more important than a small gain in social welfare.

2. Another alternative is to satisfy PE by taking the maxsum over both EF and PE. Reijnierse and Potters [1998] designed an elaborate algorithm that computes EF and PE allocations. However, these allocations are not maxsum necessarily. The techniques of Section 4.2 enable the computation of maxsum EF allocations, which are not necessarily PE. Our most important, and presumably quite challenging, open problem is finding a (tractable) algorithm that computes maxsum EF and PE allocations.

Appendix: Proof of Theorem 4.2.8

We first state the following simple propositions which shows that we can tractably compute some quantities of interest.

Proposition 1. Suppose $v_1(x) = a_1x + b_1, v_2(x) = a_2x + b_2$ on some interval I, where a_1, a_2, b_1, b_2 are k-bit rationals.

1. The intersection point of $v_1(x), v_2(x)$ can be computed in time polynomial in k, and the intersection point will be a $4k$-bit rational.

2. Let r be an $O(k)$-bit rational. The point x where $v_1(x)/v_2(x) = r$ can be computed in time polynomial in k, and this point is an $O(k)$-bit rational.

3. Let $v, w \in I$, $v < w$, v, w are $O(k)$-bit rationals.

   (a) $\int_v^w v_i(x)dx$ can be computed in time polynomial in k.

   (b) Consider δ such that $w - \delta \in I, w - \delta \geq v$. $|\int_v^w v_i(x)dx - \int_v^{w-\delta} v_i(x)dx| \leq \delta 2^{k+1}$.

Proof. 1. To compute the intersection, we set $a_1x + b_1 = a_2x + b_2$. Solving for x, we have $x = (b_2 - b_1)/(a_1 - a_2)$. It is straightforward to show that this intersection point remains a $4k$-bit rational.

2. Similar to 1., we set $(a_1x+b_1)/(a_2x+b_2) = r$ and solve for x. $x = (b_2r-b_1)/(a_1-ra_2)$.

3. (a) The result follows by observing that $\int v_i(x)dx = \int(a_ix+b_i)dx = a_ix^2/2+b_ix+C$.

   (b) The maximum value a density function can obtain using k-bit rationals is $2^k + 2^k = 2^{k+1}$. As a result, $\delta 2^{k+1}$ bounds the area under the curve over an interval of size δ.

□

Proposition 2. Consider an interval $[x_1, x_2]$, where x_1 and x_2 were either reported as an end point by the agents or the result of densities that crossed. Suppose $v_1(x) = a_1x+b_1, v_2(x) = a_2x + b_2$ on this interval, where a_1, a_2, b_1, b_2 are k-bit rationals.

1. $R_1(x)$ is either strictly increasing, strictly decreasing, or constant on $[x_1, x_2]$. If $R_1(x)$ is constant, then $R_1(x)$ is an $2k$-bit rational.

2. Suppose that $R_1(x)$ is not constant on the interval. Let $r = v_1(x)/v_2(x)$, $r' = v_1(x')/v_2(x')$. $|r - r'| \leq \delta \Rightarrow |x - x'| \leq \delta 2^{7k+1}$.

Proof. 1. The first observation is straightforward. If $R_1(x)$ is constant, then the ratio must be determined solely by the slopes of the density functions and is a_1/a_2. This is a $2k$-bit rational.

2. To prove this, we want to bound $dR_1(x)/dx$. $dR_1(x)/dx = (b_2a_1 + b_1a_2)/(a_2x + b_2)^2$. We notice that this value is maximized when the denominator is close to 0. This occurs at one of the end points of the interval. If $a_2x+b_2 = 0$ at one of the endpoints, then $a_1x + b_2 = 0$ as well at that end point (we assume that $v_1(x) \leq v_2(x)$ on the interval.) This implies that the ratio is constant on the interval, so we need not

consider that case. Therefore, $a_2x + b_2$ is strictly positive. Endpoints of the interval are $4k$-bit rationals, so the smallest possible value for the denominator occurs when $a_2 = 1/2^k, x = 1/2^{4k}$. The value at this point is $1/2^{5k}$. So we can bound $dR_1(x)/dx$ by 2^{7k+1} (a bound for the numerator of $dR_1(x)/dx$ is 2^{2k+1}, and we set the denominator to $1/2^{5k}$).

□

We now proceed to a proof of Theorem 4.2.8.

Proof sketch. We demonstrate how we can approximately implement Algorithm 1 to a given precision ϵ. Make a mark at the beginning and end of each interval as well as at 0 and 1. Let $\mathcal{J}$ be the set of intervals formed by consecutive marks. $|\mathcal{J}| = O(m)$. On each $I \in \mathcal{J}$, check whether the agents' value density functions intersect. If they do, break I into two separate intervals at the point of intersection. Each agent's value density function has constant slope on each $I \in \mathcal{J}$, so each original interval creates at most 2 new intervals, resulting in $O(m)$ intervals. By Proposition 1 the intersection points can be computed in time polynomial in k and will be $4k$-bit rationals. Let $\mathcal{J}'$ denote the new set of intervals after accounting for intersecting value density functions.

Every $I \in \mathcal{J}'$ has the following properties:

1. $v_1(x) = a_1x + b_1, v_2(x) = a_2x + b_2$ on I, for some k-bit rationals a_1, a_2, b_1, b_2.
2. Exactly one of the following holds:
   (a) $v_1(x) < v_2(x)\ \forall x \in int(I)$
   (b) $v_1(x) = v_2(x)\ \forall x \in int(I)$
   (c) $v_1(x) > v_2(x)\ \forall x \in int(I)$

With $\mathcal{J}'$ defined, we can check the different cases of Algorithm 2. We use the same notation as used in Algorithm 2, and the steps below refer to the steps in Algorithm 2. Each interval in $\mathcal{J}'$ either belongs entirely to $Y_{1\geq 2}$ or to $Y_{2>1}$. $V_1(Y_{1\geq 2})$ and $V_2(Y_{2\geq 1})$ can be computed by finding each agent's value on the appropriate intervals, applying Proposition 1:3a.

Step 1a: Allocate $Y_{1>2}$ to agent 1, $Y_{2\geq 1}$ to agent 2.
Step 1b: We need to find an allocation of $Y_{1=2}$ such that $Y_{1>2}$ along with the allocation of $Y_{1=2}$ gives agent 1 value approximately 1/2. Since $V_1(Y_{1\geq 2}) \geq 1/2$, there is some x^* such

that giving agent 1 $V_1([0, x^*] \cap Y_{1=2} \cup Y_{1>2}) = 1/2$. Let x be the multiple of $1/2^p$ that is closest to x^* without going over, with p to be specified later. x can be found by doing binary search over p-bits. Proposition 1:3b tells us that $V_1([0, x] \cap Y_{1=2} \cup Y_{1>2})$ will be within $\epsilon/2$ of $1/2$ for some $p = O(\log(1/\epsilon) + k)$, ensuring that envy is at most ϵ. Note that we always allocate intervals to an agent who weakly prefers the interval, so the allocation is optimal.

Step 2: We perform a search over the space of p_1-bit rationals (value of p_1 to be specified). We search for a value ratio in our search space that is close to and at least r^*. By stipulating that the value ratio is at least r^*, we ensure that we at least match the efficiency of the maximally efficient EF allocation since we give away fewer intervals where agent 1 has a smaller density function. Search over p_1-bit rationals can be done in $O(p_1)$ steps due to the result in Papadimitriou [1979].

The search described in Papadimitriou [1979] requires a custom subroutine given as Algorithm 3, which returns whether or not the desired value is at most some rational number r.

Algorithm 3

AtMost(r)

1. Let $u = V_1(Y_{\geq r} \cup Y_{1\geq 2}), \text{len} = V_1(Y_{>r} \cup Y_{1>2})$ (for upper and lower bound).
2. $u, \text{len} > 1/2$. Return true.
3. $u \in [1/2 - \epsilon/2, 1/2]$. Give agent 1 $Y_{\geq r}$ in addition to $Y_{1>2}$. Break.
4. $\text{len} < 1/2 < u$. Give agent 1 $Y_{>r}$. Let x be the largest p_2-bit rational such that $V_1(Y_{>r} \cup Y_{1\geq 2} \cup ([0, x] \cap Y_{=r}) \leq 1/2$. Break.
5. Return false.

AtMost can be performed tractably due to Proposition 1, which says that agent values for $Y_{\geq r}$ and $Y_{>r}$ can be computed in time polynomial in k. The final step involves showing that for appropriately chosen values of p_1, p_2, we will find an allocation that is ϵ-EF and as efficient as the maximally EF allocation. To show this, we revisit the different steps in Algorithm 2. Let $p_1 \geq 7k + 1 + \log(1/\epsilon) + \log m$ and $p_2 \geq k + 2 + \log(1/\epsilon)$.

Step 2: There are two cases:

1. $Y_{=r^*}$ contains at least one entire interval $I \in J'$. Algorithm 2 divides $Y_{=r^*}$ so that agent 1's value from $Y_{=r^*}, Y_{>r^*}, Y_{1\geq 2}$ is exactly $1/2$. Proposition 2.1 tells us that r^* is a $2k$-bit rational, so our search either finds r^* exactly or quits early because it found some r that gives agent 1 value in $[1/2 - \epsilon/2, 1/2]$. In the latter case, agent 1 is not

envious, and we have given away fewer intervals in $Y_{2>1}$ to agent 1 (compared with the optimal EF allocation). In the former case, there is some x^* such that giving agent 1 $Y_{>r^*} \cup (Y_{=r^*} \cap [0, x^*]) \cup Y_{1\geq 2}$ gives agent 1 value exactly 1/2. By searching over p_2-bit rationals, we can find an x close enough to x^* such that agent 1's value is within $\epsilon/2$ of 1/2, without going over (Proposition 1.3b). This ensures that agent 1 is EF and that efficiency is at least that of an optimal EF allocation because we allocate strictly fewer intervals where agent 1 has lower value.

2. $Y_{=r^*}$ does not contain an interval $I \in J'$. Therefore, if I is an interval with constant value ratio, $Y_{\geq r^*}$ either contains all of I or contains none of I. By Proposition 2.2, if we are close enough to r^*, then the induced x values will be closed to x^* on each $I \in J'$. By choice of p_1, we ensure that there is an r in our search space such that $V_1(Y_{\geq r} \cup Y_{1\geq 2}) \in [1/2 - \epsilon, 1/2]$. The search procedure finds such an r, and agent 1 has most ϵ envy.

□

Appendix: Proof of Theorem 4.3.7

Proof. Suppose we split the cake into three equal intervals on which agents have the following unnormalized values:

$$\text{Agent } 1 : [51, 50, 0] \tag{4.17}$$

$$\text{Agent } 2 : [50, 51, 0] \tag{4.18}$$

$$\text{Agent } 3 : [51, 10, 50] \tag{4.19}$$

Suppose we write down the primal LP constraints in vector and matrix form. The variables will be $x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32}, x_{33}$, and the constraints are ordered by 1.) interval constraints, 2.) EF constraints, ordered by agent.

Objective:

$$\begin{bmatrix} \frac{51}{101} & \frac{50}{101} & \frac{50}{101} & \frac{51}{101} & \frac{51}{111} & \frac{10}{111} & \frac{50}{111} \end{bmatrix}$$

Matrix:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{51}{101} & -\frac{50}{101} & \frac{51}{101} & \frac{50}{101} & 0 & 0 & 0 \\ -\frac{51}{101} & -\frac{50}{101} & 0 & 0 & \frac{51}{101} & \frac{50}{101} & 0 \\ \frac{50}{101} & \frac{51}{101} & -\frac{50}{101} & -\frac{51}{101} & 0 & 0 & 0 \\ 0 & 0 & -\frac{50}{101} & -\frac{51}{101} & \frac{50}{101} & \frac{51}{101} & 0 \\ \frac{51}{111} & \frac{10}{111} & 0 & 0 & -\frac{51}{111} & -\frac{10}{111} & -\frac{50}{111} \\ 0 & 0 & \frac{51}{111} & \frac{10}{111} & -\frac{51}{111} & -\frac{10}{111} & -\frac{50}{111} \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In order to make the system a system of equalities, we add slack variables for each constraint. Consider the basic feasible solution that corresponds to choosing the variables $x_{11}, x_{12}, x_{21}, x_{22}, x_{33}$ along with the slack variables for the EF constraints for $(1,3), (2,1), (2,3), (3,2)$. This results in the following solution where these variables are set as follows and all other variables are set to 0:

$$\begin{bmatrix} 133/136 & 1/80 & 3/136 & 79/80 & 1 & 1/2 & 13/680 & 693/1360 & 13/37 \end{bmatrix}$$

It remains to show that this solution is optimal. To do so, we compute the reduced costs for each variable (including the slack variables). These are:

$$[0, 0, 0, 0, -9/2960, -6077/15096, 0, -39/80, -203/408, -1195/2516, -61/4080, 0, 0, 0, -37/680, 0]$$

Since only the basic variables have non-negative reduced costs, this is the unique optimal solution. Since agent 1 obtains some of interval 2 and agent 2 obtains some of interval 1, this is not a PO solution (since we can swap) and make both agents better off. □

Chapter 5

Towards More Expressive Cake Cutting

Suppose that agents' valuation functions are piecewise uniform. Under the standard cake cutting model, agent valuations are additive. That is, agents receive value from arbitrarily small subintervals, even if the subintervals are separated from other subintervals. Using the metaphor of cake, agents are assumed to have use for "crumbs" of cake. However, there are settings where agents' valuation functions are plausibly piecewise uniform but agents have no use for these crumbs. For example, thinking of the cake as time, consider the allocation of advertising time: an agent may be interested in time slots when related shows are aired, but does not derive value from slots that are shorter than, say, thirty seconds. Similarly, one can consider priority access time to an Internet service provider for, e.g., streaming video or gaming. As a different example, consider the allocation of a strip of beach to real estate developers; each developer is interested in areas with specific properties, but has no use for tiny plots.

Therefore, more *expressiveness* is called for in the class of valuation functions. We augment piecewise uniform valuations with an option to specify the minimum length of a usable interval; we call these augmented valuations *piecewise uniform with minimum length (PUML)*. An agent's value for a piece of cake under PUML valuations is proportional to the total length of its intersection with the agent's desired intervals, excluding subintervals in the intersection that are shorter than the agent's specified minimum length.

PUML valuations depart from those used in the cake cutting literature since they can be non-additive: it may be the case that two disjoint intervals each have zero value but their union forms a contiguous interval that is longer than the minimum. In this respect, PUML

valuations blur the line between divisible and indivisible goods: some divisions of a good (i.e., an interval) are possible, whereas other divisions are impossible (as they would induce worthless subintervals). We are aware of a single existing paper that studies cake cutting under non-additive valuations [Berliant et al., 1992, Section 5], but these valuations do not include PUML valuations, and are studied only in the context of the existence of Pareto-efficient allocations. Expressiveness in mechanisms has been studied by Benisch et al. [2008] and Dütting et al. [2011], but the focal point of that work is to study the tradeoffs between simplicity and expressiveness and they assume a setting where payments are permitted. We focus on designing algorithms for our more expressive valuations.

5.1 Our Results

Consider two agents with PUML valuations, where the minimum length for each agent is 1 (that is, any strict subinterval of the entire cake is worthless). Clearly no proportional allocation exists, but an EF allocation does exist. In fact, any division that gives some nonempty piece to each agent will be EF. In such an alocation, both agents have value zero for both pieces (this shows that under PUML valuations envy-freeness does not imply proportionality even if the entire cake is allocated). Worse, one of the agents must have value zero, so even approximate proportionality in a multiplicative sense, as studied in Edmonds and Pruhs [2006a], is unattainable. We therefore consider approximate proportionality in an additive sense, to be made formal later.

In Section 5.3 we propose a polynomial-time algorithm for any number of agents with PUML valuations that provides an additive worst-case approximate proportionality guarantee. The algorithm is a generalization of a well-known fully proportional algorithm in the traditional cake cutting setting. We also prove that our algorithm is optimal, as no algorithm can attain a better worst-case proportionality guarantee.

With proportionality understood, in Section 5.4 we consider envy-freeness in combination with proportionality. For two agents with PUML valuations, we find that (rather surprisingly) we can obtain full envy-freeness while still satisfying the optimal approximate proportionality guarantee, in polynomial time. We do this via an algorithm that is very different from the approximately proportional algorithm for n agents and makes extensive use of discarding intervals in order to attain full envy-freeness.

As in the previous chapter, we do not consider self-interested agents in this chapter. In other words, our algorithm returns proportional and EF allocations with respect to the

reported valuations $V_1, \ldots, V_n$.

5.2 PUML Valuations

We augment piecewise uniform valuations with a minimum length parameter. Given $i \in N$, the minimal length parameter λ_i indicates that agent i has no value for intervals of length less than λ_i. As described in Section 3.5, agents with piecewise uniform valuations have a reference piece of cake U_i that describes the intervals the agent desires. Let $D(i, X) \equiv U_i \cap X$ describe the intervals in X desired by agent i and $d(i, X) \equiv \text{len}(D(i, X))$ describe the length of these intervals. An agent's value for a piece of cake X can be written as

$$V_i(X) = \frac{d(i, X)}{d(i, [0, 1])}.$$

In words, it is the ratio of desired lengths received to the total lengths desired by agent i.

For example, if $D(i, [0, 1]) = \{[0, 0.2], [0.5, 0.8]\}$ and $X = \{[0.1, 0.3], [0.4, 0.7]\}$, then $D(i, X) = \{[0.1, 0.2], [0.5, 0.7]\}$, $d(i, X) = 0.3$, $d(i, [0, 1]) = 0.5$, and $V_i(X) = 0.6$.

Definition 5.2.1. Under valuations that are *piecewise uniform with minimum length (PUML)*, each agent $i \in N$ uniformly desires a piece of cake $D(i, [0, 1])$, and holds a minimum length parameter λ_i. The valuation function of the agent is defined by

$$V_i(X) = \frac{\sum_{I \in D(i,X):\ |I| \geq \lambda_i} |I|}{d(i, [0, 1])}.$$

Note that the summation only includes intervals with length at least λ_i.

Going back to the previous example, if $D(i, [0, 1]) = \{[0, 0.2], [0.5, 0.8]\}$ and $X = \{[0.1, 0.3], [0.4, 0.7]\}$, and in addition $\lambda_i = 0.2$, then $\{I \in D(i, X) :\ |I| \geq \lambda_i\} = \{(0.5, 0.7)\}$, and therefore $V_i(X) = 0.4$.

We will assume that every interval I in $D(i, [0, 1])$ satisfies $|I| \geq \lambda_i$, that is, agents do not desire worthless intervals. We also assume *free disposal*, in that we can choose not to allocate part of the cake without penalty. Under PUML valuations, this assumption is without loss of generality, because we can "destroy" intervals by partitioning them into worthless tiny subintervals. Even under PUML valuations, when we write $D(i, X)$, we refer to $U_i \cap X$, even if some of these intervals might have length less than λ_i. Similarly, $d(i, X)$ refers to the length of the desired intervals that intersect X, ignoring the minimum length requirement.

In order to discuss computational complexity, as we do below, we must understand how

the input is represented. PUML valuations can be concisely represented via the boundaries of the desired intervals, and $\lambda_1, \ldots, \lambda_n$. The size of the input is the number of bits used to represent these parameters. As we have been assuming in this thesis, we assume the direct revelation model where agents report their entire valuation function to the algorithm.

5.3 Proportionality

As discussed in Section 3.4.1, under standard assumptions on agent valuations, the Dubins-Spanier moving knife procedure proves that proportional allocations always exist. Under PUML valuations, proportionality (or even multiplicative approximate proportionality) is not always achievable, as demonstrated by the example given earlier in this chapter where both agents desire the entire cake and $\lambda_1 = \lambda_2 = 1$. Therefore, we seek to achieve an additive approximate proportionality guarantee. This guarantee will depend on λ_i: agents with larger λ_i are guaranteed less, as having a larger λ_i restricts the allocations that can give the agent its proportionality guarantee. Let $\ell_i = \lambda_i / d(i, [0,1])$ for each $i \in N$.

Definition 5.3.1. An allocation $X_1, \ldots, X_n$ is *β-proportional* with respect to valuations $V_1, \ldots, V_n$ if for all $i \in N$, $V_i(X_i) \geq 1/n - \beta \cdot \ell_i$. A direct revelation cake cutting algorithm is *β-proportional* if when given input $V_1, \ldots, V_n$, it always produces an allocation that is β-proportional with respect to $V_1, \ldots, V_n$.

Equivalently, a β-proportional algorithm guarantees that

$$\sum_{I \in D(i, X_i):\ |I| \geq \lambda_i} |I| \geq \frac{d(i, [0,1])}{n} - \beta \cdot \lambda_i.$$

5.3.1 Algorithmic Results

To achieve approximately proportional allocations, we present Algorithm 4. This algorithm is inspired by the Dubins-Spanier procedure, but shifts the points where agents metaphorically say "stop" in a way that, as we shall see, provides optimal guarantees. The output of the algorithm is the allocation $X_1, \ldots, X_n$, which is fully assigned before the algorithm returns.

This is obviously a polynomial-time algorithm. The following theorem quantifies its proportionality guarantees.

Theorem 5.3.2. *Under PUML valuations, Algorithm 4 is* $(2(n-1)/n)$ *-proportional and polynomial-time.*

Algorithm 4 $(2(n-1)/n)$-proportional algorithm

Input: $V_1, \ldots, V_n$

1. PROPALLOCATE$(N, 0, (V_1, \ldots, V_n))$

PROPALLOCATE$(S, u, (V_1, \ldots, V_n))$:

1. If $S = \{i\}$, set $X_i = [u, 1]$ and return.
2. For each $i \in S$:
   $r_i^* = \min\{r : r \in [u,1], V_i([u,r]) \geq \frac{V_i([u,1])}{|S|} - \frac{2(|S|-1)\ell_i}{|S|}\}$.
3. $r^* = \min_{i \in S} r_i^*, i^* = \arg\min_{i \in S} r_i^*$ (break ties arbitrarily).
4. Set $X_{i^*} = [u, r]$.
5. PROPALLOCATE$(S \setminus \{i^*\}, r)$

In particular, the algorithm is at most 1-proportional for $n = 2$. Before proving the theorem, we establish a simple lemma.

Lemma 5.3.3. *Consider a contiguous interval $[u, v]$. Let $w \in [u, v]$. Then $V_i([w, v]) \geq V_i([u, v]) - V_i([u, w]) - 2\ell_i$.*

Proof. $V_i([u, w]) + V_i([w, v])$ can be smaller than $V_i([u, v])$ because the cut point w might break an interval that was previously of length at least λ_i into two intervals that have length less than λ_i. For instance, suppose $w \in (b, c)$ where $(b, c) \in D_i([u, v])$. If $w - b \geq \lambda_i, c - w \geq \lambda_i$, then no value is lost by adding the values for $[u, w], [w, v]$ separately compared to the value for $[u, v]$. However, if $w - b < \lambda_i$ or $c - w < \lambda_i$, then we lose value by adding values over $[u, w], [w, v]$ separately. The most value that can be lost is $2\ell_i$ (ℓ_i on either side of the cut at w). $V_i([u, w]) + V_i([w, v]) \geq V_i([u, v]) - 2\ell_i$, and rearranging yields the lemma. □

Proof of Theorem 5.3.2. The proof proceeds by induction on $|S|$ and showing that PROPALLOCATE gives each $i \in S$ at least value $V_i([u, 1])/|S| - 2(|S| - 1)\ell_i/|S|$, i.e., PROPALLOCATE is $2(|S| - 1)/|S|$- proportional with respect to $[u, 1]$ and the agents in S. This is straightforward to show for the allocated agent i^*. For the remaining agents, we use the fact that the agents were not chosen in combination with Lemma 5.3.3 to conclude that they have a large enough value for the unallocated cake such that the inductive hypothesis provides their proportionality guarantee. □

5.3.2 Impossibility Results

Consider once more the case of two agents. In this case Algorithm 4 guarantees that each agent receives value of $1/2 - \ell_i$. Suppose that both agents desire the entire cake and $\lambda_1 = \ell_1 = \lambda_2 = \ell_2 = 1/2 + \epsilon$; then one agent will receive value of zero, that is, it is impossible

to guarantee a value of more than $1/2 - \ell_i + \epsilon$ for any $\epsilon > 0$; hence the algorithm is optimal for the case of $n = 2$. More generally, the following theorem establishes that the algorithm is worst-case optimal for any number of agents.

Theorem 5.3.4. *For every n there exist PUML valuations such that no cake cutting algorithm is β-proportional for $\beta < 2(n-1)/n$.*

Proof. Let $\epsilon = \epsilon(n) < 1/n$, and consider $n-1$ disjoint subintervals of length ϵ. Assume each agent desires each of the $n-1$ subintervals. Set $\lambda_i = \epsilon/2 + \delta$ for $\delta > 0$ and all $i \in N$. Let x be the linear multiplier of ℓ_i that specifies the guarantee for each agent. Clearly in any allocation there is at least one agent $i \in N$ that gets zero in desired intervals, that is,

$$\frac{1}{n} - x\frac{\lambda_i}{d(i, \{0,1\})} \leq 0.$$

Substituting, we get

$$\begin{aligned} & \frac{1}{n} - x\frac{\epsilon/2 + \delta}{(n-1)\epsilon} \leq 0 \\ \Rightarrow \quad & \frac{n-1}{n} \cdot \epsilon - x\left(\frac{\epsilon}{2} + \delta\right) \leq 0. \end{aligned}$$

Rearranging yields

$$x \geq \frac{2(n-1)}{n} - \frac{2\delta}{\frac{\epsilon}{2} + \delta} \cdot \frac{n-1}{n} \geq \frac{2(n-1)}{n} - \frac{4\delta}{\epsilon}.$$

The theorem follows by taking $\delta \to 0$. □

Theorem 5.3.4 does not exclude the possibility that, for a given instance, there is an allocation with a better degree of proportionality than the one computed by Algorithm 4. Indeed, there could be an algorithm that matches Algorithm 4 in the worst case but returns allocations that have better proportionality guarantees in other cases. However, the combinatorial richness of PUML valuations imposes limits on what can be achieved via *polynomial-time* algorithms due to the following "inapproximability result."

Theorem 5.3.5. *For any constant $\epsilon \in (0, 1/2)$, given n agents with PUML valuations such that $\ell_i < 1/n$, it is NP-hard to distinguish between the following two statements: (a) there is a $(1/2 + \epsilon)$-proportional allocation and (b) no $(3/2 - \epsilon)$-proportional allocation exists.*

Proof (Sketch). We reduce the 3-dimensional matching problem to deciding whether an instance of PUML valuations has an (1/2+ϵ)-proportional allocation or no (3/2-ϵ)-proportional

allocation exists. A full proof is deferred to the appendix of this chapter. □

In particular, for a given set of valuations $V_1, \ldots, V_n$, let $\gamma(V_1, \ldots, V_n)$ be the smallest value γ such that a γ-proportional allocation exists. Theorem 5.3.5 says that there is no polynomial time algorithm that always returns a $(\gamma(V_1, \ldots, V_n) + 1 - 2\epsilon)$-proportional allocation for every set of valuations (if there were then we could solve the NP-hard problem in the statement of Theorem 5.3.5). In other words, their is no polynomial-time algorithm that approximates the best proportional allocation within an additive factor of 1. Algorithm 1 is close to optimal when viewed under this measure since it provides an additive $2(n-1)/n$ approximation guarantee.

5.4 Proportionality and Envy-Freeness

While Theorem 5.3.2 provides an optimal worst-case proportionality guarantee, it does not address envy-freeness. In fact, it is possible for an agent to be the first allocated yet have greater value for pieces later allocated to other agents.

A natural question to ask is whether we can attain envy-freeness while satisfying the proportionality guarantee of Algorithm 4. In other words, is there an algorithm that is $2(n-1)/n$-proportional and fully EF? For the case of two agents, we show that indeed there is an algorithm (that is 1-proportional and EF), and we leave open the challenging question of a general number of agents.

Under general classic valuations, finding an EF and proportional allocation is trivial when $n = 2$ as we can use the Cut and Choose algorithm However, immediate variations of this algorithm do not yield envy-freeness under PUML valuations. In fact, we will see that achieving envy-freeness and 1-proportionality under PUML valuations is surprisingly difficult. Our solution makes extensive use of the free disposal assumption, which was not required above, in order to attain envy-freeness.

We introduce some new notation that is specific to this section. Lengths of intervals will be denoted by Greek letters. It will be convenient to refer to disjoint subintervals of a given interval. We define a *filtering* F to be a function that takes an interval and returns a set of disjoint subintervals of the given interval. For instance, for the interval $[0, 0.25]$, we might choose to only allocate $\{[0, 0.1], [0.2, 0.25]\}$, throwing away $[0.1, 0.2]$. In this case, $F([0, 0.25]) = \{[0, 0.1], [0.2, 0.25]\}$.

5.4.1 An Algorithmic Skeleton

We first observe that if a filtering with specific properties exists, then a 1-proportional and EF allocation exists for two agents.

Definition 5.4.1. A filtering, point pair (F_i, x_i) is *fair* if

$$V_i(F_i([0, x_i])) = V_i(F_i([x_i, 1])) \tag{5.1}$$

$$V_i(F_i([0, x_i])) \geq 1/2 - \ell_i \tag{5.2}$$

Assuming that we can find fair filtering, point pair, Algorithm 5 is 1-proportional and EF. The high level idea is to start with a feasible allocation is that 1-proportional based on the fair filtering, point pair. If some agent is envious, then let that agent choose between the pieces generated by the other agent's fair filtering, point pair.

Algorithm 5 1-proportional and EF algorithm for $n = 2$

1. Compute fair filtering and point pairs (F_1, x_1), (F_2, x_2).
2. Assume $x_1 \leq x_2$. Otherwise, the roles can be reversed.
3. Let $X_1 = F_1([0, x_1])$, $X_2 = F_2([x_2, 1])$. If this is EF, return.
4. If both agents are envious, then swap the allocations and return.
5. If agent 1 is envious, let agent 1 choose between $F_2([0, x_2])$ and $F_2([x_2, 1])$, giving agent 2 the piece that was not chosen.
6. If agent 2 is envious, let agent 2 choose between $F_1([0, x_1])$ and $F_1([x_1, 1])$, giving agent 1 the piece that was not chosen.

Lemma 5.4.2. *Assume that there exist fair filtering, point pairs (F_1, x_1), (F_2, x_2). Then Algorithm 5 is* 1*-proportional and EF.*

Proof. If Algorithm 5 terminates at Step 3, then both agents receive their proportionality guarantee by (5.2) and are not envious. If Algorithm 5 terminates at Step 4, there is no envy since both agents preferred the other's initial allocation. The proportionality guarantee is satisfied since each agent prefers its final allocation to its initial, but the initial allocation satisfies (5.2). If Algorithm 5 terminates at Step 5, agent 2 receives its proportionality guarantee, is indifferent, and so is not envious. Agent 1 receives its proportionality guarantee since it at worst receives $F_2([x_2, 1])$ which it preferred to $F_1([0, x_1])$. Since agent 1 gets to choose between $F_2([0, x_2])$ and $F_2([x_2, 1])$, agent 1 cannot be envious. A similar argument applies to termination at Step 6. □

If we can carry out Step 1 of Algorithm 5, that is, compute fair filtering, point pairs (F_1, x_1), (F_2, x_2), then Lemma 5.4.2 allows us to find a 1-proportional and EF allocation. We show that such filtering, point pairs always exist (and can be computed efficiently) in the next section.

5.4.2 Finding Fair Filtering, Point pairs

Algorithm 5 implies that we can treat the agents independently, as long as for any v_i and λ_i we can find a filtering, point pair F_i, x_i. Therefore, we drop the agent subscripts and pretend we are dealing with a single agent.

Before proceeding to the main constructive proof, we establish a result about what lengths in desired intervals are attainable by throwing away intervals in the allocation.

Lemma 5.4.3. *Suppose $d(X) = k\lambda + \epsilon$ for some positive integer k and $0 \leq \epsilon < \lambda$, i.e., the length of the agent's desired intervals on X total $k\lambda+\epsilon$. It is possible to throw away intervals in X so that the agent receives desired lengths worth exactly $k\lambda + \epsilon_1$ for any $0 \leq \epsilon_1 \leq \epsilon$. It is also possible to throw away intervals in X so that the agent receives exactly $(k-1)\lambda$ in desired lengths.*

Proof. (a). Let $0 \leq \epsilon_1 \leq \epsilon$. If there is some desired interval that has length greater than 2λ, then we can remove $\epsilon - \epsilon_1$ in lengths from one side of the interval. If no interval has length greater than 2λ, then each interval has length in $[\lambda, 2\lambda)$. The sum of the excess above λ must be at least ϵ, so we can remove lengths of $\epsilon - \epsilon_1$ without decreasing any interval to less than λ in length. (b) By (a), we can attain desired lengths of exactly $k\lambda$. If any remaining interval has length exactly λ, then we can remove that interval. Otherwise, there is at least λ in excess that can be removed without decreasing any interval to length less than λ. □

We now proceed to the main proof that a fair filtering, point pair (F, x) always exists. Let c be the center of the cake from the point of view of the agent, i.e. $d([0, c]) = d([c, 1]) = d([0, 1])/2$. Note that there may be an infinite number of such points, so we take the right-most one.

Let y and z denote the left and right end points of the desired interval that contains c, as seen in Figure 5.1.

Although the agent may not receive value $1/2$ from $[0, c]$ and $[c, 1]$ because $c - y$ and $z - c$ may be smaller than λ, $[0, c]$ and $[c, 1]$ always satisfy the proportionality guarantee for both agents. This is formalized in the following Lemma.

Lemma 5.4.4. $V([0, c]) \geq 1/2 - \ell, V([c, 1]) \geq 1/2 - \ell$.

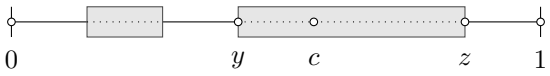


Figure 5.1: Desired intervals are shaded. The points y and z are the left and right boundaries of the interval containing c.

Proof. The proof is immediate by observing that each agent can lose at most λ in value on either side of the cut point c. □

In the case where $V([0,c]) = V([c,1])$, the identity filtering along with c is a fair filtering, point pair.

We therefore assume $V([0,c]) < V([c,1])$. To construct a fair filtering, point pair (F,x), we choose an $x \in \{y,c,z\}$ based on certain conditions. We then apply Lemma 5.4.3 to throw away intervals in $[0,x],[x,1]$ until the agent is indifferent between the two intervals. Symmetric arguments can be applied to handle the case $V([0,c]) > V([c,1])$.

We consider two separate cases, based on whether $z - c < \lambda$ or $z - c \geq \lambda$. Since $V([0,c]) < V([c,1]) \leq 1/2$, we also have $c - y < \lambda$.

Case I: $z - c < \lambda$**.**

Let $\delta = c - y, \gamma = z - c$, as depicted in Figure 5.2.

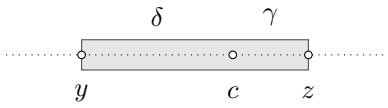


Figure 5.2: The case where $z - c < \lambda$.

From above, we have that $c - y < \lambda$, so $\delta < \lambda$. Therefore, $V([0,c]) = V([0,y]) = 1/2 - \delta/d([0,1])$ and (since $\gamma < \lambda$) $V([c,1]) = 1/2 - \gamma/d([0,1])$. Since $V([0,c]) < V([c,1])$, we obtain that $\gamma < \delta$.

Suppose that the desired lengths in $[z,1]$ and $[0,y]$ are uniquely expressed as $k\lambda + \rho$ and $k_2\lambda + \rho_2$, respectively, where k, k_2 are positive integers and $0 \leq \rho, \rho_2 < \lambda$. Since $V([0,c]) < V([c,1])$, $k_2\lambda + \rho_2 < k\lambda + \rho$. First, we show that $k_2 \geq k - 1$. Given the way we chose c, $k_2\lambda + \rho_2 + \delta = k\lambda + \rho + \gamma$. Therefore, $(k - k_2)\lambda = \rho_2 - \rho + \delta - \gamma$. Since the variables on the right hand side are all non-negative and $\rho_2 < \lambda, \delta < \lambda$, $(k - k_2)\lambda < 2\lambda$ so $k_2 \geq k - 1$. Now there are two cases:

Case I.1: $k_2 = k$, therefore $\rho > \rho_2$. Set $x = c$ and, using Lemma 5.4.3a, we throw away intervals in $[z,1]$ so that the agent receives desired lengths $k_2\lambda + \rho_2$ in both $[0,x]$ and $[x,1]$.

Case I.2: $k_2 = k - 1$. First observe that $\gamma + \delta \geq \lambda$ (otherwise, the agent would not have desired this interval). Set $x = z$ and throw away $[y + \lambda, z]$. The agent receives desired lengths $k\lambda + \rho_2$ and $k\lambda + \rho$ in $[0, x]$ and $[x, 1]$, respectively. If $\rho \geq \rho_2$, by Lemma 5.4.3a, we throw away intervals in $[x, 1]$ so that the agent receives desired lengths exactly $k\lambda + \rho_2$ in both $[0, x]$ and $[x, 1]$. If $\rho < \rho_2$, by Lemma 5.4.3a, we throw away intervals in $[0, x]$ so that the agent receives desired lengths exactly $k\lambda + \rho$ in both $[0, x]$ and $[x, 1]$.

In both cases, 1-proportionality is satisfied as the agent always receives at least $k_2\lambda + \rho_2$ in desired intervals and therefore at least its value for $[0, c]$. By Lemma 5.4.4 we know that this satisfies 1-proportionality.

Case II: $z - c \geq \lambda$**.**

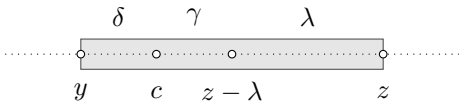


Figure 5.3: The case where $z - c \geq \lambda$.

$V([c, 1]) = 1/2$ since the agent does not lose desired lengths in $[c, 1]$. Let $\delta = c - y, \gamma = z - \lambda - c$, as seen in Figure 5.3. It is easy to handle the case where $\gamma \geq \delta$ by throwing away interval $[c, c + \delta]$ and by setting $x = c$. The crucial observation is that since $\gamma \geq \delta$, no desired lengths are lost in $[c+\delta, 1]$ and, hence, the agent has desired lengths of $d([0, 1])/2-\delta$ in both $[0, x]$ and $[x, 1]$. In the following, assume $\gamma < \delta$. $\delta < \lambda$ since we assume that $V([0, c]) < V([c, 1]) = 1/2$.

Let $[0, y]$ and $[z-\lambda, 1]$ provide $k_2\lambda+\rho_2$ and $k\lambda+\rho$ in desired lengths for positive integers k, k_2 and $0 \leq \rho, \rho_2 < \lambda$. As in the previous case, we can show that $k_2 \geq k - 1$ by using the fact that $k_2\lambda + \rho_2 + \delta = \gamma + k\lambda + \rho$. The details are omitted since they are the same as in the previous case. We can now distinguish between two cases:

Case II.1: $k_2 = k$. Since $\gamma < \delta$ and $V([0, c]) < V([c, 1])$, $\rho \geq \rho_2$. Set $x = c$, throw away interval $[c, z - \lambda]$ and use Lemma 5.4.3a to throw away intervals in $[z - \lambda, 1]$ so that the agent gets exactly $k\lambda + \rho_2$ in both $[0, x]$ and $[x, 1]$.

Case II.2: $k_2 = k - 1$. Then, the interval $[z, 1]$ gives desired lengths of $(k - 1)\lambda + \rho$. First, use Lemma 5.4.3b to throw away lengths in $[z, 1]$ so that the desired lengths in $[z, 1]$ are exactly $(k - 2)\lambda$. Set $x = y$. If $\rho_2 \leq \gamma + \delta$, throw away interval $[y, z - \lambda - \rho_2]$ in order to obtain desired lengths of $(k - 1)\lambda + \rho_2$ in both $[0, x]$ and $[x, 1]$. If $\rho_2 > \gamma + \delta$, use Lemma

5.4.3a to throw away lengths in $[0, x]$ so that the desired lengths in $[0, x]$ and $[x, 1]$ are exactly $(k-1)\lambda + \gamma + \delta$. This is the only case in which the desired lengths in $[0, x]$ and $[x, 1]$ are less than $V([0, c])$. So, we still need to prove that $(k-1)\lambda + \gamma + \delta$ satifies the proportionality requirement (5.2). By the definition of point c, we have $(k-1)\lambda + \rho_2 + \delta = \gamma + k\lambda + \rho = d([0,1])/2$, i.e., $\rho_2 = \lambda + \rho + \gamma - \delta$ which implies that $\delta > \rho + \gamma$ since $\rho_2 < \lambda$. Hence, $(k-1)\lambda + \gamma + \delta > (k-1)\lambda + \rho + 2\gamma \geq d([0,1])/2 - \lambda$.

5.4.3 Tying Things Together

Now that we have proven the existence of fair (F, x) for any agent valuations, we can apply Lemma 5.4.2. In addition, note that Section 5.4.2 implicitly provides a computationally efficient implementation of Step 1 of Algorithm 5, so the algorithm is clearly polynomial-time. We therefore have the following result.

Theorem 5.4.5. *Assume that $n = 2$ and the agents have PUML valuations. Then Algorithm 5 is 1-proportional, envy-free, and polynomial-time.*

5.5 Discussion

Normalization

Though we have presented this chapter assuming that agent valuations are normalized, the positive results still hold even if agent valuations are not normalized. The main difference is that the proportionality guarantees now scale with the agent's total value for $[0, 1]$. If an agent has total value α for $[0, 1]$, then the proportionality guarantee becomes $\alpha/n - \alpha \cdot \beta \cdot \ell_i$. The negative results trivially still hold since we allow a larger set of agent valuations.

5.6 Summary and Future Work

In this chapter, we consider valuations that are more expressive than standard piecewise uniform valuations. Specifically, we allow agents to have a minimum length parameter which indicates that they have no value for intervals less than a certain length. This captures the possibility that agents do not gain value for very small, disjoint intervals of the divisible good being allocated. The main results are an algorithm for finding approximately proportional allocations (and a theorem showing that this result is tight) and an algorithm for finding approximately proportional and EF allocations for two agents.

5.6.1 Future Work

1. A positive side effect of our algorithmic framework is that it encourages agents not to be "greedy": the smaller an agent's λ_i is, the larger the degree of proportionality the agent is guaranteed. We wish to emphasize though that this is not a formal game-theoretic statement. Indeed, under the algorithms presented in this paper, agents can certainly gain by lying about their valuation function or even about their minimum length. In contrast, in the next chapter, we design a fully proportional, EF algorithm that is also truthful under piecewise uniform valuations (without minimum length). There is a large gap, both conceptual and technical, between piecewise uniform and PUML valuations. It would be interesting to know whether there is a $(2(n-1)/n)$-proportional and *truthful* algorithm under PUML valuations.

2. Algorithm 4, which works for any number of agents, is inspired by a proportional algorithm that works for all valuations functions under classic assumptions. Similarly, in a sense Algorithm 5 extends the Cut and Choose algorithm. Unfortunately, in general envy-freeness is hard to obtain for more than two agents (see Section 3.4.1), and in particular the techniques of Algorithm 5 do not appear to generalize to any number of agents. Progress on this front would require fundamentally new techniques.

Appendix: Proof of Theorem 5.3.5

Proof. Sketch of proof. The reduction uses a trick from an old paper on scheduling of Lenstra, Shmoys, and Tardos (Mathematical Programming, 1990).

We start with an instance of the NP-hard 3-dimensional matching problem consisting of three sets A, B, and C with m elements each and a collection of q triplets of the form (a_i, b_j, c_k) where a_i, b_j, and c_k belong to A, B, and C, respectively. The question is whether there exist m triplets that cover all the elements or not.

We call triplets that contain a_j triplets of type j. Let t_j be their number. Let δ be a negligibly small positive number. We have the following disjoint intervals:

- A triplet interval of length $3 + \delta$ for each triplet.
- Many additional intervals of length $3 - \delta$. Let L be their number.

We have the following agents:

- For each element of $B \cup C$, there is an element agent. Such an agent has value for the triplet intervals corresponding to a triplet to which its corresponding element belongs and all additional intervals. Its minimum length requirement is 1.

- For each $j = 1, ..., m$, there are $2t_j - 1$ type agents. Each such agent has value for the triplet intervals corresponding to triplets of type j and the additional intervals. Its minimum length requirement is $1 + \delta$.

- Many auxiliary agents. Their exact number is twice the number of additional intervals. These agents have value for the additional intervals only. Their minimum length requirement is $1 + \delta$.

Note that the total number of agents is exactly $3m + 2(qm) + 2L$ and the number of intervals is $q + L$. By setting L to a (polynomially) large number compared to m and q, we have that the exact proportionality requirement for each agent is slightly below 3/2.

Observe that one triplet interval can accommodate either at most two element agents (the ones corresponding to the two elements of B and C that are contained in the triplet) and at most one type agent (the one with type equal to the type of the triplet), or (at most) two type agents (and no element agents). Also, one additional interval can accommodate at most two element, type, or auxiliary agents.

Now, if the original instance has a perfect matching then there exists an allocation in which each agent has non-zero value (either 1 or $1 + \delta$ depending on whether it is element, type, or auxiliary agent). The element agents take their triplet interval corresponding to the triplet of the perfect matching, the type agents ll the space in triplet intervals of their type, and the auxiliary agents occupy the additional intervals. This is an almost 1/2-proportional allocation. Otherwise, this means that, there are at most $m - 1$ intervals that accommodate three agents, so some agent should receive no value, i.e., amost 3/2-proportional allocation. □

Chapter 6

Truthful Cake Cutting

Thus far we have discussed various cake cutting problems that emerge once we adopt a direct revelation model with restricted valuation functions. In Chapter 4 we examine maxsum fair allocations, both in terms of computing these allocations and understanding the properties of different maxsum fair allocations. In Chapter 5, we present a slight modification to the classic cake cutting model and piecewise uniform valuations that allows for more expressiveness. We did not consider strategic issues in either chapter, but rather assumed access to true information on agent valuations.

In this chapter, we view cake cutting as a mechanism design problem where payments are not permitted. Our goal is to find cake cutting mechanisms (or algorithms) that are DSIC. Very little prior work considers strategic issues in cake cutting, and the work that does examines a very weak notion of truthfulness. Specifically, prior work considers an algorithm to be truthful if there *exist* valuations of other agents such that reporting truthfully is a strict best response [Brams et al., 2006, 2008]. In contrast, DSIC requires that *for all* valuations of other agents reporting truthfully weakly dominates any other report. Additionally, this earlier work also operates in the classic cake cutting model. To make things precise, for the rest of this chapter, we refer to mechanisms that satisfy the notion of truthfulness in prior work as *weakly-truthful* and we refer to DSIC mechanisms as *truthful*.

As a concrete illustration of the difference between the two notions, consider the most basic cake cutting mechanism for the case of two agents, the Cut and Choose mechanism introduced in Section 3.4.1. The mechanism as described in Section 3.4.1 is described with the agents taking actions; equivalently, in a direct revelation approach, the mechanism can act on behalf of agents using the reported valuations. Cut and Choose is weakly truthful, since if agent 1 divides the cake into two pieces that are unequal according to its valuation

then agent 2 may prefer the piece that is worth more to agent 1. Agent 2 clearly cannot benefit by lying. However, the mechanism is not truthful (in the DSIC sense). Indeed, consider the case where agent 1 would simply like to receive as much cake as possible, whereas the single-minded agent 2 is only interested in the interval $[0, \epsilon]$ where ϵ is small (for example, it may only be interested in the cherry). If agent 1 follows the protocol it would only receive half of the cake. Agent 1 can do better by reporting that it values the intervals $[0, \epsilon]$ and $[\epsilon, 1]$ equally.

In this chapter we consider the design of truthful and fair (proportional and EF) cake cutting mechanisms. We restrict ourselves to considering agents with piecewise uniform valuations and operate under a direct revelation model. It is trivial to find proportional and EF allocations in this setting; the richness of our problem stems from our desire to additionally achieve truthfulness.

6.1 Our Results

In Section 6.4 we consider deterministic mechanisms when agent valuations are piecewise uniform. Our main result is a deterministic mechanism for any number of agents that is truthful, proportional, EF, Lorenz dominant in a certain sense, Pareto-efficient, and polynomial-time when agents have piecewise uniform valuations.

To gain intuition for our mechanism, it is insightful to examine the special case of two agents. In the two agent mechanism, the crux of the allocation is how we should allocate intervals that both agents desire. We can throw away intervals that are undesired by any agent, and allocate the intervals that are only desired by a single agent to that agent. Our mechanism then allocates the intervals desired by both agents to try to equalize the total lengths given to each agent. If the length of intervals only desired by agent i exceeds the length of intervals only desired by agent j, then agent i will be given a smaller share of the mutually desired intervals. To see how this encourages truthfulness, consider the deviation where an agent misstates and increases the lengths of intervals only desired by that agent. The agent receives all of these intervals (which include intervals it does not actually desire), but obtains a smaller share of the mutually desired intervals. It turns out that the equalization of lengths removes the incentive for agents to misstate their valuations.

In the general mechanism for n agents, our mechanism examines what the fairness requirement means for the allocation given to different agents. Agents who receive a small amount of the cake due to fairness requirements are handled first and given the best possible

fair allocation. As an example of agents who might be handled first, there could be a set of agents who all desire the same small interval of the cake. For fairness to be maintained, each of these agents must receive the same fraction of this small interval. These agents would be selectively handled earlier by the mechanism. By protecting the disadvantaged agents in this way, the mechanism removes the incentive to report larger intervals than those that are truly desired. Our mechanism is highly dependent on the assumption that agents hold piecewise uniform valuations and there is no easy way to generalize our techniques to piecewise constant valuations.

In Section 6.5 we consider randomized mechanisms. We slightly relax truthfulness by asking that the mechanism be *truthful in expectation*, that is, an agent cannot hope to increase its expected value by lying for any reports of other agents. For general valuations, we present a simple randomized mechanism that is truthful in expectation, and is *ex-post* proportional and EF. Our result relies on the existence of *perfect partitions*, which are partitions of the cake where every agent has value exactly $1/n$ for every element of the partition. Given a perfect partition, the randomized mechanism constructs an allocation by indexing the elements of the partition, and allocating the elements to agents based on a randomly drawn permutation. The perfect partition guarantees fairness, while truthfulness in expectation is preserved because the agent's expected value is a function of the agent's value for the entire cake and not of the specific partition. While perfect partitions are known to exist, there is no known algorithm for constructing perfect partitions for general valuations. We provide explicit constructions of perfect partitions when agents hold piecewise linear valuations.

6.2 Related Work

In independent parallel work, Mossel and Tamuz [2010] ask similar questions about truthful and fair cake cutting. However, their work only considers randomized mechanisms, and they provide existence results rather than concrete mechanisms for specific classes of valuations. They also focus solely on proportionality rather than proportionality and envy-freeness together. Under general assumptions on valuation functions (which are essentially the same as our definition of valid valuations), they show that there exists a mechanism that is truthful in expectation and always guarantees each agent a value of more than $1/n$. The results are then extended to the case of indivisible goods. The technical overlap between the two papers is very small; we refer the reader's attention to this overlap in a footnote in

Section 6.5.

Thomson [2007] showed that a truthful and Pareto-efficient mechanism must be dictatorial in the slightly different setting of pie-cutting. In pie cutting, the pie is modeled as a circular object and the feasible cuts are wedges, while in cake cutting the cake is modeled as the interval [0,1] and feasible cuts are subintervals. Though the two settings appear similar and even possibly equivalent, they are actually distinct and results for one setting do not readily carry over to the other. Brams et al. [2008] provide a more in depth discussion. Note that Pareto-efficiency is not a fairness property and neither implies, nor is implied by, envy-freeness or proportionality.

Our deterministic mechanism is closely related to the *egalitarian solution* studied by Dutta and Ray [1989] for cooperative games and Bogomolnaia and Moulin [2004] for the random assignment problem with dichotomous preferences. It is also related to the *probabilistic serial* mechanism studied by Bogomolnaia and Moulin [2001] and Katta and Sethuraman [2006] for the random assignment problem. However, the cake cutting setting is distinct from the cooperative games setting and the random assignment problem, so though our mechanism uses similar ideas and techniques, our results are not implied by these earlier results. The proof of strategyproofness in Bogomolnaia and Moulin [2004] does carry over to our setting (when interpreted through the correct lens), but we provide our independently discovered proof in the sequel.

Dutta and Ray [1989] propose the *egalitarian solution* for cooperative games with transferable utility. As cake cutting is not a transferable utility setting, the connection to Dutta and Ray [1989] involves defining a convex, cooperative game when agents have piecewise uniform valuations in the cake cutting setting. Our deterministic mechanism finds an allocation that is closely related to the egalitarian solution, and our algorithm for finding the solution also resembles that proposed in this earlier work. We expand on this connection in Section 6.4.5, where we use it to show that our deterministic mechanism finds an allocation that is Lorenz dominant in a certain sense.

Our mechanism also resembles the mechanisms proposed by Bogomolnaia and Moulin [2001], Katta and Sethuraman [2006], and Bogomolnaia and Moulin [2004] for the random assignment problem under different assumptions on preferences. Bogomolnaia and Moulin [2001] study the setting where agents have strict ordinal preferences, Katta and Sethuraman [2006] extend this to the full preference domain (where indifferences are allowed), and Bogomolnaia and Moulin [2004] study the case when agents have dichotomous preferences (i.e., items are either acceptable or unacceptable and agents are indifferent between any two

acceptable items).

Because we study cake cutting when agents have piecewise uniform valuations, we can relate the cake cutting setting to the random assignment setting as follows. Given agents' reports, make a mark at the beginning of and end of each agent's desired intervals. Construct an item for every subinterval formed by consecutive marks. Giving an agent an item with probability p is the same as allocating the agent a p-fraction of the associated subinterval. While this transformation converts cake cutting into a discrete allocation problem, there are fundamental differences between our setting and the random assignment setting. First, in our setting, agents' valuations affect the items that are created, whereas in the random assignment problem items are assumed to be given. Second, in our setting, agents want as many items as possible, while the random assignment setting assumes that each agent wants at most one item. In addition to these two general differences, there are differences specific to each of the previously studied settings.

If we look specifically at dichotomous preferences studied by Bogomolnaia and Moulin [2004], our agents may not have dichotomous preferences for items created based on agent reports; that is, some subintervals might be larger than others and agents, though they have piecewise uniform valuations, might not have the same value for all desired items. Even if we discretize the cake into many tiny items (to try and equalize agent values for desired items), our agents still desire as many of the created items as possible. Despite these differences, Theorem 2 of Bogomolnaia and Moulin [2004] does carry over to our setting if we view the allocation matrices Z as specifying assignments of agents to intervals of $[0, 1]$. We discovered this relationship after publication of our results, and we describe this in more detail in Section 6.4.5.

If we look specifically at strict ordinal preferences studied by Bogomolnaia and Moulin [2001], our agents may be indifferent between the items formed by subintervals between consecutive marks. On the other hand, our agents cannot hold arbitrary ordinal preferences over subintervals between consecutive marks, since if two agents desire two subintervals, both agents would value the longer subinterval more than the shorter. This last difference also persists under the full preference domain studied by Katta and Sethuraman [2006]. Because of these differences, our results cannot be derived from these earlier results, despite the underlying dependence on similar ideas.

The network flow ideas that we use to prove properties of our mechanism are similar to those used by Katta and Sethuraman [2006] to show that the egalitarian assignment solution can be computed in polynomial time. Interestingly, Katta and Sethuraman [2006]

note that the egalitarian assignment solution is identical to another independent mechanism for finding a lexicographically optimal flow in a network due to Megiddo [1979].

Recently, after the initial publication of our work, Maya and Nisan [2012] consider the two agent case where agents have piecewise uniform valuations. They characterize truthful mechanisms in this setting and derive bounds on the welfare of truthful mechanisms in comparison to the optimal social welfare. Interestingly, they show that the truthful mechanisms that have the best worst-case guarantees on social welfare are the truthful and fair mechanisms, which are equivalent to our deterministic mechanism applied to two agents.

6.3 Preliminaries

A cake cutting mechanism f is *truthful* if when an agent lies it is allocated a piece of cake that is worth, according to its real valuation, no more than the piece of cake it was allocated when reporting truthfully. Formally, denote $A_i = f_i(V_1, \ldots, V_n)$, and let $\mathcal{V}$ be a class of valuation functions. The mechanism f is truthful if for every agent i, every collection of valuations functions $V_1, \ldots, V_n \in \mathcal{V}$, and every $V_i' \in \mathcal{V}$, it holds that $V_i(f_i(V_1, \ldots, V_n)) \geq V_i(f_i(V_1, \ldots, V_{i-1}, V_i', V_{i+1}, \ldots, V_n))$.

Our results require that agent valuations be elements of a restricted family of valuation functions. For our deterministic mechanism, we require that agent valuations are piecewise uniform. For our randomized mechanism, we allow agent valuations to be piecewise linear.

6.4 Deterministic Mechanisms

In order to attain a truthful, deterministic mechanism, we focus on the restricted family of piecewise uniform valuations. The restricted family is still rich enough to make the problem challenging and capture real world situations, and we believe that our results provide a foundation for future work on truthful cake cutting. Indeed, it remains an open question whether truthful, deterministic mechanisms exist for even the slightly richer family of piecewise constant valuations. In the rest of this section, we assume that valuations are piecewise uniform.

6.4.1 A Deterministic Mechanism

Before introducing our mechanism we present some required notation. Let $S \subseteq N$ be a subset of agents and let X be a piece of cake. Let $D(S, X)$ denote the portions of X that

are valued by at least one agent in S. Formally, recalling that U_i is the reference piece of cake for agent i (the piece that specifies the agents' desired intervals), $D(S, X) = \left(\bigcup_{i \in S} U_i\right) \cap X$, and is itself a union of intervals.

Let $\text{avg}(S, X) = \text{len}(D(S, X))/|S|$ denote the average length of intervals in X desired by at least one agent in S. We say that an allocation is *exact* with respect to S and X if it allocates to each agent in S a piece of cake of length $\text{avg}(S, X)$ comprised *only* of desired intervals. Clearly this requires allocating all of $D(S, X)$ since the total length of allocated intervals is $\text{avg}(S, X) \cdot |S| = \text{len}(D(S, X))$. Suppose $S = \{1, 2\}$ and $X = [0, 1]$: if $U_1 = U_2 = [0, 0.2]$ then agents 1 and 2 receiving $[0, 0.1]$ and $[0.1, 0.2]$ respectively is an exact allocation; but if $U_1 = [0, 0.2], U_2 = [0.3, 0.7]$ then there is no exact allocation. since agent 1 cannot be given 0.3 in desired lengths.

The deterministic mechanism for n agents with piecewise uniform valuations is a recursive mechanism that finds a subset of agents with a certain property, makes the allocation decision for that subset, and then makes a recursive call on the remaining agents and the remaining intervals. Specifically, for a given set of agents $S \subseteq N$ and a remaining piece of cake to be allocated X, we find the subset $S' \subseteq S$ of agents with the smallest $\text{avg}(S', X)$. We then give an exact allocation of $D(S', X)$ to S'. We recurse on $S \setminus S'$ and the intervals not desired by any agent in S', i.e. $X \setminus D(S', X)$. The pseudocode of the mechanism is given as Mechanism 6.

Mechanism 6 $(V_1, \ldots, V_n)$

1. FairAllocate$(\{1, \ldots, n\}, [0, 1], (V_1, \ldots, V_n))$

FairAllocate$(S,\ X,\ V_1, \ldots, V_n)$:

1. If $S = \emptyset$, return.
2. Let $S_{\min} \in \underset{S' \subseteq S}{\text{argmin}}\ \text{avg}(S', X)$ (breaking ties arbitrarily).
3. Let $E_1, \ldots, E_n$ be an exact allocation with respect to $S_{\min}, X$ (breaking ties arbitrarily). For each $i \in S_{\min}$, set $A_i = E_i$.
4. FairAllocate$(S \setminus S_{\min}, X \setminus D(S_{\min}, X), (V_1, \ldots, V_n))$.

In particular, Steps 2 and 3 of FairAllocate imply that if $S = \{i\}$ then $A_i = D(S, X)$. For example, suppose $X = [0, 1]$, $U_1 = [0, 0.1]$, $U_2 = [0, 0.39]$, and $U_3 = [0, 0.6]$. In this case, the subset with the smallest average is $\{1\}$, so agent 1 receives all of $[0, 0.1]$ and we recurse on $\{2, 3\}, [0.1, 1]$. In the recursive call, set $\{2\}$ has average $0.39 - 0.1 = 0.29$, set $\{3\}$ has average $0.6 - 0.1 = 0.5$, and set $\{2, 3\}$ has average $(0.6 - 0.1)/2 = 0.25$. As a result, the entire set $\{2, 3\}$ is chosen as the set with smallest average, and an exact allocation of

$[0.1, 1.0]$ is given to agents 2 and 3. One possible allocation is to give agent 2 $[0.1, 0.35]$ and agent 3 $[0.35, 0.6]$. Note that if agent 1 uniformly values $[0, 0.2]$ instead, the first call would choose $\{1, 2\}$ as the subset with the smallest average, equally allocating $[0, 0.39]$ between agents 1 and 2 and giving the rest, $[0.39, 0.6]$, to agent 3.

Our goal in the rest of this section is to prove the following theorem.

Theorem 6.4.1. *Assume that the agents have piecewise uniform valuation functions. Then Mechanism 6 is truthful, proportional, EF, Lorenz dominant (in a certain sense), Pareto-efficient, and polynomial-time.*

To prove the theorem we exploit a connection to network flow in Section 6.4.3. As explained in Section 6.2, this technique is a simple generalization of related results in the economics and operations research literature, but we include some of the details for completeness. Our main technical contributions in this section are the truthfulness and fairness of Mechanism 6, which are established in Section 6.4.5.

6.4.2 The Two Agent Mechanism

To gain intuition for the general case of Theorem 6.4.1 we first describe the special case of two agents. Note that designing truthful, proportional and EF mechanisms even for this case is nontrivial. To see the difficulty, consider an intuitive first attempt at a proportional and EF mechanism. We have already seen that Cut and Choose is not truthful for two agents. Another straightforward approach would be to mark the end points of all submitted intervals and divide every resulting subinterval equally between the two agents. One possibility is to always give the left half of each subinterval to agent 1 and the right half to the agent 2. This mechanism is clearly proportional and EF since every agent receives value exactly $1/2$. However, it is not truthful due to a simple example. Under this mechanism, if both agents value the entire cake, agent 1 receives $[0, 0.5]$ and the agent 2 receives $[0.5, 1]$. Suppose that agent 1's true valuation consists of only $[0, 0.5]$. If it reports truthfully, it receives $[0, 0.25], [0.5, 0.75]$ which gives value 0.5. The agent can gain by instead reporting that it values all of $[0, 1]$ and receive $[0, 0.5]$ which gives it value 1. In particular, suppose that when both agents report $[0, 1]$, agent 1 receives $[0, 0.5]$. In order for the mechanism to be truthful, whenever agent 1 reports some set of subintervals of $[0, 0.5]$ and agent 2 reports $[0, 1]$, the mechanism must allocate to agent 1 all of this agent's desired intervals.

We now discuss our truthful Mechanism 6 for the case of two agents. Recall that U_i denotes the reference piece of cake of agent $i \in N$, and for $i \in \{1, 2\}$ let $W_i = U_i \setminus U_{3-i}$, and

$W_{12} = U_1 \cap U_2$. W_1, W_2, W_{12} are disjoint, and this will allow us to interchangeably write $\text{len}(W_1) + \text{len}(W_2) + \text{len}(W_{12})$ and $\text{len}(W_1 \cup W_2 \cup W_{12})$.

For two agents, the decision of the mechanism hinges on the choice of $S_{\min}$ in the first call to FairAllocate. There are three possibilities for $S_{\min} : \{1\}, \{2\}, \{1, 2\}$. If $S_{min} = \{i\}$, then agent i receives $U_i = W_i \cup W_{12}$ and agent $3 - i$ receives $U_{3-i} \setminus U_i = W_{3-i}$. On the other hand, if $S_{min} = \{1, 2\}$, then the mechanism gives an exact allocation with each agent receiving equal lengths. Agent i receives all of W_i and W_{12} is split so that the total lengths for each agent are equal.

Properties of the two agent mechanism

To prove that the mechanism for two agents is well-defined, fair, Pareto-efficient, and truthful,[7] it is useful to introduce variables $\delta_i = (\text{len}(W_{12}) - \text{len}(W_i) + \text{len}(W_{3-i}))/2$ for each agent. It holds that $\delta_1 + \delta_2 = \text{len}(W_{12})$ and $\delta_1 + \text{len}(W_1) = \text{len}(W_{12} \cup W_1 \cup W_2)/2 = \delta_2 + \text{len}(W_2)$. Qualitatively, δ_i measures the share of W_{12} that agent i is entitled to due to the size of W_i. Smaller values of $\text{len}(W_i)$ give the agent a larger claim to W_{12} and increase δ_i. We can relate values of these variables to the choices of $S_{\min}$ made by the mechanism.

By definition of W_i and W_{12}, $\text{avg}(\{1\}) = \text{len}(W_1 \cup W_{12})$, $\text{avg}(\{2\}) = \text{len}(W_2 \cup W_{12})$, $\text{avg}(\{1, 2\}) = \text{len}(W_1 \cup W_2 \cup W_{12})/2$. There are three cases:

1. $\delta_1 < 0, \delta_2 > \text{len}(W_{12})$. Since $\text{len}(W_1) + \delta_1 = \text{len}(W_2) + \delta_2$, $\text{len}(W_1) > \text{len}(W_2)$ and therefore $\text{avg}(\{1\}) > \text{avg}(\{2\})$. Moreover, using $\delta_2 > \text{len}(W_{12})$, we have

$$\frac{\text{len}(W_{12}) - \text{len}(W_2) + \text{len}(W_1)}{2} > \text{len}(W_{12}),$$

   and it follows that

$$\frac{\text{len}(W_{12}) + \text{len}(W_2) + \text{len}(W_1)}{2} > \text{len}(W_{12}) + \text{len}(W_2),$$

   that is, $\text{avg}(\{1, 2\}) > \text{avg}(\{2\})$.

   This case corresponds to $S_{min} = \{2\}$, and the mechanism gives all of $W_2 \cup W_{12}$ to agent 2 and all of W_1 to agent 1.

2. $0 \leq \delta_1 \leq \text{len}(W_{12}), 0 \leq \delta_2 \leq \text{len}(W_{12})$. We can use the same arguments as the previous case with the inequalities flipped to show that $\text{avg}(\{1, 2\}) < \text{avg}(\{1\})$ and

[7] We defer a discussion of Lorenz dominance to the proof for the general case with n agents.

$\text{avg}(\{1,2\}) < \text{avg}(\{2\})$. This case corresponds to $S_{min} = \{1,2\}$, and the mechanism gives an exact allocation of $W_1 \cup W_2 \cup W_{12}$ to the agents.

3. $\delta_2 < 0, \delta_1 > \text{len}(W_{12})$. The analysis is the same as case 1, with agents 1 and 2 changing roles. This corresponds to $S_{min} = \{1\}$. Agent 1 receives all of $W_1 \cup W_{12}$ and agent 2 receives all of W_2.

First, we show that Mechanism 6 is well-defined. Mechanism 6 calls for an exact allocation to be made to S_{min} at each step, but this may not always be possible. When $|S_{min}| = 1$ an exact allocation is trivial, but when $S_{min} = \{1,2\}$, we need to guarantee that we can exactly split $D(\{1,2\},[0,1])$. Because $S_{min} = \{1,2\}$ only when $0 \leq \delta_1, \delta_2 \leq \text{len}(W_{12})$, this is possible by giving W_i and δ_i of W_{12} to agent i.

Next, we establish the fairness properties of proportionality and envy-freeness. If $S_{min} = \{1,2\}$, proportionality and envy-freeness are clear. If $S_{min} = \{1\}$, then agent 1 receives all of its desired intervals so its value is 1. Agent 2 receives only W_2, but since $S_{min} = \{1\}$, $\text{len}(W_1 \cup W_{12}) \leq \text{len}(W_1 \cup W_{12} \cup W_2)/2$ which implies $\text{len}(W_{12}) \leq \text{len}(W_2)$, and hence agent 2 receives value at least $1/2$ and is not envious of agent 1.

Pareto-efficiency is easy to see because in all cases, we allocate W_i to agent i and fully allocate W_{12}, so it is not possible to give more lengths to one agent without decreasing the allocation of the other.

In order to establish Theorem 6.4.1 for the two agent case it remains to show truthfulness. We will take the point of view of agent $i \in \{1,2\}$. If it holds that $\delta_i > \text{len}(W_{12})$ then the agent receives all desired intervals and has no incentive to deviate, hence we can assume that $\delta_i \leq \text{len}(W_{12})$.

Note that an agent always receives all of W_i, so profitable manipulations will try to obtain more of W_{12} by increasing δ_i while not losing too much of W_i. Also note that by definition of δ_i, an increase or decrease of $\text{len}(W_i)$ by k will respectively decrease or increase δ_i by $k/2$.

Suppose that agent i reports $U_i' \neq U_i$, inducing new pieces W_{12}', W_1', W_2'. Before manipulating the agent receives $\text{len}(W_i) + \max(\delta_i, 0)$. Note that

$$\text{len}(W_{12}') + \text{len}(W_{3-i}') = \text{len}(U_{3-i}) = \text{len}(W_{12}) + \text{len}(W_{3-i}),$$

that is, this sum is not affected by the report of agent i.

If $\text{len}(W_1') = \text{len}(W_1)$ then δ_i is unchanged, and the agent receives the same length of intervals (though the actual intervals received may not be desired). If $\text{len}(W_i') = \text{len}(W_i) - k$

then the agent loses at least k of W_i and gains at most $k/2$ from an increase in δ_i; this is not profitable. Finally, if $\text{len}(W_i') = \text{len}(W_i) + k$, then the agent gains undesired intervals of length k and δ_i is weakly smaller, so the agent does not gain any more of W_{12} by deviating.

An interpretation of the two agent mechanism.

When $|S| = 2$, the mechanism is equivalent to a swapping procedure. The proof of this equivalence is postponed to the appendix of this chapter. As before, let W_1, W_2, W_{12} describe the intervals that only agent 1 desires, only agent 2 desires, and both agents desire. Discard the intervals that neither agent desires, and give an initial grant of half of W_1, W_2, W_{12} to each agent. Assume without loss of generality that $\text{len}(W_1) \leq \text{len}(W_2)$. The swapping procedure can be described as follows.

1. Swap pieces X, Y of equal length where agent 1 owns X, agent 2 owns Y, $X \subseteq W_2$, $Y \subseteq W_1$.
2. Swap pieces X, Y of equal length where agent 1 owns X, agent 2 owns Y, $X \subseteq W_2$, $Y \subseteq W_{12}$.
3. If there are still pieces of W_2 owned by agent 1, give these intervals to agent 2.

This procedure first gives each agent an equal piece of each kind of interval. The first swaps performed are mutually beneficial: each agent gives pieces of the other agent's desired intervals in exchange for pieces of its own desired intervals. However, eventually, agent 1 obtains all of W_1 and these swaps no longer exist. The second swaps involve trades where agent 2 receives remaining pieces of W_2 in exchange for giving up pieces of W_{12}. This trade does not improve the utility of agent 2, but does improve the utility of agent 1. If agent 2 gives away all of its share of W_{12}, then agent 1 receives all of W_1 and W_{12}, and agent 2 is given all of W_2 due to step 3 which gives the remaining shares of W_2 to agent 2 for free. If agent 2 obtains all of W_2 without relinquishing all of its share of W_{12}, then agents 1 and 2 split W_{12} (potentially unequally), and each agent i receives W_i.

6.4.3 Exact Allocations and Maximum Flows

Having defined Mechanism 6 and shown that it has the desired properties in the case of two agents, we now generalize the proofs to n agents. Before turning to properties of truthfulness and fairness, we point out that, as in the two agent case, it is unclear whether Mechanism 6 is well-defined. In particular, the mechanism requires an exact allocation E with respect to

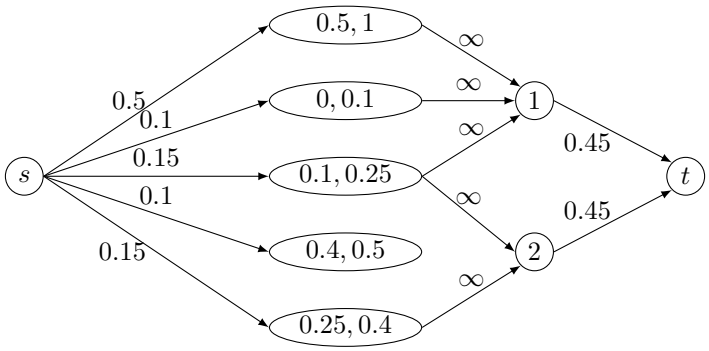


Figure 6.1: The flow network induced by the example.

the subset $S_{\min}$ and X, but it remains to show that such an allocation exists, and to provide a way to compute it. To this end we exploit a close relationship between exact allocations and maximum flows in networks.

For a given set of agents $S \subseteq N$ and a piece of cake to be allocated X, define a graph $G(S, X)$ as follows. We keep track of a set of marks, which will be used to generate nodes in $G(S, X)$. First mark the left and right boundaries of all intervals that are contained in X. For each agent $i \in N$ and subinterval in U_i, mark the left and right boundaries of subintervals that are contained in $U_i \cap X$. When we have finished this process, each pair of consecutive markings will form an interval such that each agent either uniformly values the entire interval or values none of the interval. In $G(S, X)$, create a node for each interval I formed by consecutive markings, and add a node for each agent $i \in N$, a source node s, and a sink node t. For each interval I, add a directed edge from source s to I with capacity equal to the length of the interval. Each agent node is connected to t by an edge with capacity $\text{avg}(S, X)$. For each interval-agent pair (I, i), add a directed edge with infinite capacity from node I to the agent i if agent i desires interval I.

For example, suppose $U_1 = [0, 0.25] \cup [0.5, 1]$ and $U_2 = [0.1, 0.4]$. If $X = [0, 1]$ then the interval markings will be $\{0, 0.1, 0.25, 0.4, 0.5, 1\}$. Agent 1 values $[0, 0.1]$, both agents value $[0.1, 0.25]$, agent 2 values $[0.25, 0.4]$, neither agent values $[0.4, 0.5]$ and agent 1 values $[0.5, 1]$. It holds that $\text{len}(D(\{1, 2\}, [0, 1])) = 0.9$. Average values are 0.75, 0.3 and 0.45 for sets $\{1\}, \{2\}$ and $\{1, 2\}$ respectively. See Figure 6.1 for an illustration of the induced flow network.

Lemma 6.4.2. *Let $S \subseteq N$, and let X be a piece of cake. There is a flow of size $\text{len}(D(S, X))$ in $G(S, X)$ if and only if for all $S' \subseteq S$, $\text{avg}(S', X) \geq \text{avg}(S, X)$.*

We prove the lemma using an application of the classic Max-Flow Min-Cut Theorem

(see, e.g., Cormen et al. [2001]).

Proof. Assume that for all $S' \subseteq S$, $\text{avg}(S', X) \geq \text{avg}(S, X)$. By the Max-Flow Min-Cut Theorem, the minimum capacity removed from a graph in order to disconnect the source and sink is equal to the size of the maximum flow. The only edges with finite capacity in $G(S, X)$ are the ones that connect agent nodes to the sink, and the ones that connect the source to the interval nodes.

Construct a candidate minimum cut by disconnecting some set of agent nodes $T \subseteq S$ from the sink at cost $|T| \cdot \text{avg}(S, X)$ and then disconnecting all the (s, I) connections to interval nodes I desired by an agent $i \in S \setminus T$. This means that the total additional capacity we need to remove is $\text{len}(D(S \setminus T, X))$, the total length of intervals desired by at least one agent in $S \setminus T$. By assumption, this is at least $|S \setminus T| \cdot \text{avg}(S, X)$. As a result, this cut has capacity of at least

$$|T| \cdot \text{avg}(S, X) + |S \setminus T| \cdot \text{avg}(S, X) = |S| \cdot \text{avg}(S, X) = \text{len}(D(S, X)).$$

In the other direction, assume that there is a flow of size $\text{len}(D(S, X))$ in $G(S, X)$, and assume for contradiction that there exists $S' \subseteq S$ such that $\text{avg}(S', X) < \text{avg}(S, X)$. Construct a cut by disconnecting the (s, I) connections to interval nodes desired by an agent $i \in S'$, and disconnecting the agent nodes $S \setminus S'$ from the sink. The total capacity of the cut is

$$\begin{aligned} |S'| \cdot \text{avg}(S', X) + |S \setminus S'| \cdot \text{avg}(S, X) &< |S'| \cdot \text{avg}(S, X) + |S \setminus S'| \cdot \text{avg}(S, X) \\ &= |S| \cdot \text{avg}(S, X) \\ &= \text{len}(D(S, X)), \end{aligned}$$

and by the Max-Flow Min-Cut Theorem the maximum flow must be of size less than $\text{len}(D(S, X))$, in contradiction to our assumption. □

The following lemma establishes that a flow of size $\text{len}(D(S, X))$ in $G(S, X)$ induces an exact allocation.

Lemma 6.4.3. *Let $S \subseteq N$, and let X be a piece of cake. There exists an exact allocation with respect to S, X if and only if there exists a maximum flow of size $\text{len}(D(S, X))$ in $G(S, X)$.*

Proof. Suppose that we have a maximum flow of size $\text{len}(D(S, X))$; we show how to use this flow to generate an exact allocation with respect to S, X. For each edge between interval node I and agent $i \in N$ that receives positive flow of c in the max flow, allocate c of I to agent i. This allocation is feasible because the interval nodes represent disjoint subintervals of X and the flow to the interval's node is limited by the capacity of the edge between s and the interval node, which is the length of the interval. In addition, this allocation must give each agent exactly $\text{avg}(S, X)$ in desired intervals. To see this, note that all paths to the sink must pass through agent nodes, and the sum of capacities of the edges between between the agents and the sink is $D(S, X)$. For a maximum flow to have size $D(S, X)$, these edges must saturated.

In the other direction, suppose that we have an exact allocation with respect to S, X. We can generate a feasible flow of size $\text{len}(D(S, X))$ by setting a flow of c on an edge (I, i) if agent i receives c of interval I in the exact allocation, and saturating all the edges (s, I) and (i, t) for intervals I and agents $i \in N$. □

By combining the "if" directions of Lemma 6.4.2 and Lemma 6.4.3 we see that the mechanism is indeed well-defined: if S has the smallest average then there exists an exact allocation with respect to S, X. The network in Figure 6.1 does not satisfy the minimum average requirement and does not provide a corresponding exact allocation. Moreover, we obtain a tractable mechanism for computing an exact allocation, by computing the maximum flow and deriving an exact allocation. A maximum flow can be computed in time that is polynomial in the number of nodes, that is, polynomial in our input size (see, e.g., Cormen et al. [2001]).

6.4.4 Polynomial Time

In order to show that Mechanism 6 can be implemented in polynomial time it remains to show that it is also possible to implement Step 2 of FairAllocate in polynomial time. Indeed, an efficient implementation of Step 2 would mean that FairAllocate can be implemented in polynomial time, and there are at most $n + 1$ calls to FairAllocate. So, the task is to find $S_{min} \in \text{argmin}_{S' \subseteq S} \text{avg}(S, X)$ in polynomial time, given $S \subseteq N$ and a piece of cake X. This can be done using network flow arguments, which are an easy variation on those employed by Katta and Sethuraman [2006] (see Section 6.2 for a full discussion). For completeness we quickly describe an implementation that is less efficient than Katta and Sethuraman [2006] but nevertheless polynomial time.

Given $S \subseteq N$, a piece of cake X, and $c > 0$, construct a graph $G'(S, X, c)$; this graph is identical to $G(S, X)$ as defined above, except that the capacity on the edges between the agents and the sink is c (instead of $\text{avg}(S, X)$). The proof of the following statement is identical to the proof of Lemma 6.4.2 (by replacing $\text{avg}(S, X)$ with c everywhere): there is a flow of size $c|S|$ in the network $G'(S, X, c)$ if and only if for all $S' \subseteq S$, $\text{avg}(S', X) \geq c$.

Assume that the boundaries of the agents' reference pieces of cake are represented by at most k bits, i.e. multiples of $1/2^k$. For ease of exposition (so that we deal with integers rather than rationals), scale the interval by $2^k n!$ so that the boundaries are an element of $\{0, n!, 2n!, \ldots, 2^k n!\}$.

There is a maximum c^* such that $G'(S, X, c^*)$ has a flow of size $c^*|S|$. This c^* is a member of a set of numbers that includes all the possible values of $\text{avg}(S', X)$ for $S' \subseteq S$. Because we have scaled the boundaries, the possible values for the average are contained in $\{0, 1, \ldots, 2^k n!\}$. Indeed, the numerator is an element of $\{0, n!, 2n!, \ldots, 2^k n!\}$, and the denominator is in $\{1, \ldots, n\}$. It holds that we can search for c^* by performing binary search over $\{0, 1, \ldots, 2^k n!\}$ in polynomial time. Binary search takes $O(k + n \log n)$ iterations, with each iteration taking time polynomial in k, n. Even though we multiply values by $2^k n!$, the number of bits needed is only $O(k + n \log n)$ and still polynomial in n, k.

Now, consider $c^* + 1$. This network does not have a network flow of size $(c^* + 1)|S|$. We can find a minimum cut in this network in polynomial time (see, e.g., [Cormen et al., 2001]). In this cut there is a subset $T \subseteq S$ such that the cut separates $S \setminus T$ from the sink, and the intervals desired by T from the source. This subset T must have $\text{avg}(T, X) < c^* + 1$, and since $\text{avg}(T, X)$ (after scaling) must be an integer, $\text{avg}(T, X) \leq c^*$.

We wish to claim that T is the S_{min} we are looking for. Indeed, note that since there is a flow of size $c^*|S|$ in $G'(S, X, c^*)$, it must hold that for all $S' \subseteq S$, $\text{avg}(S', X) \geq c^*$, which directly implies the claim.

6.4.5 Fairness, Efficiency, Truthfulness

Our main tool in proving that Mechanism 6 is truthful and EF is the following lemma.

Lemma 6.4.4. *Let $S_1, \ldots, S_m$ be the ordered sequence of agent sets with the smallest average as chosen by Mechanism 6 and $X_1, \ldots, X_m$ be the ordered sequence of pieces to be allocated in calls to* FairAllocate. *That is, $X_1 = [0, 1], X_2 = X_1 \setminus D(S_1, X_1), \ldots, X_m = X_{m-1} \setminus D(S_{m-1}, X_{m-1})$. Then for all $i > j$, $avg(S_i, X_i) \geq avg(S_j, X_j)$, and agents that are members of later sets receive weakly more in desired lengths.*

Proof. Suppose not. Then at some point, $\text{avg}(S_i, X_i) > \text{avg}(S_{i+1}, X_{i+1})$. Now consider $S_i \cup S_{i+1}$. We will show that $\text{avg}(S_i \cup S_{i+1}, X_i) < \text{avg}(S_i, X_i)$, contradicting the choice of S_i as the subset of agents with the smallest avg at step i. Note that S_i and S_{i+1} are disjoint since agents are removed once they have been part of the subset with the smallest average. Thus,

$$\begin{aligned}
\text{avg}(S_i \cup S_{i+1}, X_i) &= \frac{\text{len}(D(S_i \cup S_{i+1}, X_i))}{|S_i| + |S_{i+1}|} \\
&= \frac{\text{len}(D(S_i, X_i)) + \text{len}(D(S_{i+1}, X_i \setminus D(S_i, X_i)))}{|S_i| + |S_{i+1}|} \\
&= \frac{|S_i| \cdot \text{avg}(S_i, X_i) + |S_{i+1}| \cdot \text{avg}(S_{i+1}, X_{i+1})}{|S_i| + |S_{i+1}|} \\
&< \frac{|S_i| \cdot \text{avg}(S_i, X_i) + |S_{i+1}| \cdot \text{avg}(S_i, X_i)}{|S_i| + |S_{i+1}|} \\
&= \text{avg}(S_i, X_i),
\end{aligned}$$

where the second transition is true since S_i and S_{i+1} are disjoint, and the inequality follows from our assumption. □

Envy-freeness

Envy-freeness now follows immediately from Lemma 6.4.4. Indeed, consider an agent $i \in N$, and as before let S_j be the subset of agents with the smallest average in the j'th call to FairAllocate. Suppose $i \in S_j$. The agent does not envy other agents in S_j since these agents are given an exact allocation and all receive the same length in desired intervals. By Lemma 6.4.4, the agent does not envy agents in S_k for $k < j$ because the amount agents receive weakly increases with each call. The agent does not envy agents in S_k for $k > j$ because all intervals desired by the agent are allocated and removed from consideration when the agent receives its allocation.

Proportionality

Before we establish proportionality, we prove the following useful lemma, which shows that no intervals are wasted in the allocation produced by Mechanism 6.

Lemma 6.4.5. *Mechanism 6 assigns all subintervals desired by at least one agent to an agent who desires the interval.*

Proof. Every interval in $[0, 1]$ is either allocated in one of the calls to FairAllocate, or it is left unallocated by the final call to FairAllocate. If an interval is allocated in one of

the calls to FairAllocate, then it is part of an exact allocation, and in an exact allocation, each allocated agent receives only desired intervals. If an interval is left unallocated in the final call to FairAllocate, then it must be that no agent desired the interval. Indeed, all intervals desired by agents allocated prior to the final call have already been allocated, and the final call allocates all of $D(S', X')$ to agents in S', where S' and X' are the inputs to the final call of FairAllocate. □

Combining Lemma 6.4.5 with envy-freeness gives us proportionality. Indeed, suppose that some agent i receives less than $\text{len}(D(\{i\}, [0,1])/|S|$ of its desired intervals. Because no desired intervals are thrown away, some other agent must receive a length of at least $\text{len}(D(\{i\}, [0,1])/|S|$ of the desired intervals of agent i, in contradiction to envy-freeness.

Lorenz Dominance

The allocation produced by Mechanism 6 is related to the egalitarian solution for supermodular transferable utility cooperative games described by Dutta and Ray [1989]. In particular, the length in intervals received by an agent in Mechanism 6 is equal to the value assigned to an agent in the egalitarian solution of a corresponding convex cooperative game.

To translate our setting into the cooperative game setting, consider the following coalitional value function for each set of agents S:

$$v(S) = \text{len}(D(S, [0,1]))$$

This value function is clearly concave, so to translate to the convex setting, we consider the negation. Viewed in this light, we have a cost-sharing scenario, where the interval $[0,1]$ represents inputs to production which must be purchased. For instance, we can think of $[0,1]$ as a strip of land, and each agent requires irrigation canals on certain subintervals of this strip of land. Once irrigation canals are built on a given subinterval, they can be used jointly by all the agents.

For convex games, Dutta and Ray [1989] give an algorithm (which is very similar to Mechanism 6) to find the egalitarian solution, and they show that the egalitarian solution Lorenz dominates every solution in the core of the cooperative game. To understand the significance of this result in our setting, we first need to understand Lorenz dominance and the core.

Translated to our setting, Lorenz dominance is the following criterion. For a given allocation $A_1, \ldots, A_n$, define a lengths vector $\ell(A_1, \ldots, A_n)$ which sorts the lengths in intervals

agents receive from smallest to largest. An allocation $A_1, \ldots, A_n$ Lorenz dominates an allocation $A'_1, \ldots, A'_n$ if $\sum_{i=1}^{k} \ell_i(A_1, \ldots, A_n) \geq \sum_{i=1}^{k} \ell_i(A'_1, \ldots, A'_n)$ for all $1 \leq k \leq n$ with strict inequality for some k. In other words, the partial sums of the lengths vector are weakly greater, with strict inequality for at least one partial sum. As an example, if we have two agents that split the interval [0,1] (they each receive lengths of 0.5), then this allocation Lorenz dominates an allocation that gives the entire interval to a single agent.

Translated to our setting, the core consists of all allocations where no subset of agents S is allocated more than $\text{len}(D(S, [0, 1]))$ in intervals. In particular, the core contains at least all allocations that do not allocate intervals to agents who do not desire them. Indeed, if agents only receive desired intervals, there is no way that the total length of intervals received by agents in a subset S can exceed $\text{len}(D(S, [0, 1])$.

Given the correspondence between the egalitarian solution and the allocation produced by Mechanism 1, the allocation Lorenz dominates any other allocation that only allocates intervals to agents who desire them and has a distinct lengths vector. There may be multiple allocations with the same lengths vector. As a result, the allocation we produce does not Lorenz dominate every other allocation; however, it does Lorenz dominate every other allocation that generates a distinct lengths vector.

It was *a priori* unclear whether there would exist an allocation in which the desired lengths received by agents corresponds to the values in the egalitarian solution because our setting is not a transferable utility setting. This is what is established in Section 6.4.3. Moreover, our definition of Lorenz dominance looks at the lengths vector and not agent utilities. Indeed, it is easy to show that a similar criterion is not satisfied if we characterize allocations by the vectors of agent's utilities.

Pareto-efficiency

Each agent's utility is a function of the length in desired intervals it receives. As a result, Pareto-efficiency follows directly from the discussion on Lorenz dominance.

Truthfulness

We establish truthfulness by examining the possible ways in which an agent can affect the progression of Mechanism 6. In particular, we examine whether an agent $i \in S$ would want to deviate from truthfulness in each call to FairAllocate(S, X). Let S_{min} denote the set of agents with the smallest average if agent i reports truthfully. For a given call to FairAllocate, agent i can report a valuation which changes the exact allocation made to

S_{min} or which changes the choice of S_{min} (or both).

We first consider deviations which try to modify the exact allocation for S_{min}, X. The exact allocation for S_{min} only depends on the reports of agents in S_{min}, so we assume $i \in S_{min}$. If agent i reports truthfully, it receives exactly $\text{len}(D(S_{min}, X))/|S_{min}|$ in desired lengths. Agent i can deviate, causing the set of desired intervals to change to $D'(S_{min}, X)$. If $\text{len}(D'(S_{min}, X)) \leq \text{len}(D(S_{min}, X))$, then this is not profitable. Suppose that $\text{len}(D'(S_{min}, X)) > \text{len}(D(S_{min}, X))$. Now each agent receives $\text{len}(D'(S_{min}, X))/|S_{min}|$ of intervals. The intervals received by agents other than i are in $D(S_{min}, X)$ since the other agents have not changed their reports. Subtracting off these intervals, the maximum length in desired intervals agent i can receive from this deviation is $\text{len}(D(S_{min}, X)) - (|S| - 1) \cdot \text{len}(D'(S_{min}, X))/|S_{min}| \leq \text{len}(D(S_{min}, X))/|S_{min}|$, and this deviation is not profitable.

We next consider deviations which attempt to change the choice of subset with the smallest average. There are two cases to consider. Let avg' denote the new averages induced by a deviation of agent i.

1. $i \notin S_{min}$. For any set S', if $i \notin S'$, agent i cannot change $D(S', X)$ with its reports. As a result, agent i cannot make the mechanism choose some other $S', i \notin S'$ since the agent cannot change $\text{avg}(S_{min}, X)$ or $\text{avg}(S', X)$. Therefore, the agent's only deviation is to try and make the mechanism choose a set $S', i \in S'$. However, in order to do so, the agent must make $\text{avg}'(S', X) \leq \text{avg}(S_{min}, X)$ and will receive $\text{avg}'(S', X)$ in desired lengths. By Lemma 6.4.4, the agent was receiving at least $\text{avg}(S_{min}, X)$ under truthful reports since the agent was chosen in a later round, so this is not profitable.

2. $i \in S_{min}$. Agent i receives $\text{avg}(S_{min}, X)$ if it reports truthfully. Suppose agent i deviates and forces selection of another set S' as the set with the smallest average. If $i \in S'$, then to be profitable, $\text{avg}'(S', X) \geq \text{avg}(S_{min}, X)$. If $i \notin S'$, then agent i could not have affected $\text{avg}(S', X)$ and S' was not chosen when i was truthful, so $\text{avg}'(S', X) \geq \text{avg}(S_{min}, X)$. Either way, to be profitable, it must be that $\text{avg}'(S', X) \geq \text{avg}(S_{min}, X)$. Now consider some agent $j \in S_{min}$. When agent i reported truthfully, agent j was receiving $\text{avg}(S_{min}, X)$ of $D(S_{min}, X)$. If $j \in S'$, then agent j receives exactly $\text{avg}'(S', X)$ in intervals. If $j \notin S'$, then by Lemma 6.4.4, agent j receives at least $\text{avg}'(S', X)$ in intervals. In either case, agent j receives at least $\text{avg}'(S', X)$ in intervals.

   Using arguments similar to the case above, we can now show that no deviation is profitable. While agent i deviates, the other agents in S_{min} do not, so the intervals

they receive are a part of $D(S_{min}, X)$. This leaves

$$\begin{aligned} & \text{len}(D(S_{min}, X)) - (|S_{min}| - 1) \cdot \text{avg}'(S', X) \\ \leq\ & \text{len}(D(S_{min}, X)) - (|S_{min}| - 1) \cdot \text{avg}(S_{min}, X) \\ =\ & \text{avg}(S_{min}, X) \end{aligned}$$

in desired intervals for agent i, so the deviation is not profitable.

This completes the proof of truthfulness. Putting everything together gives us Theorem 6.4.1.

Group Strategyproofness

Group strategyproofness is a stronger notion than truthfulness that requires that it is not possible for any group of agents to collectively deviate and make each of their allocations weakly better off while making at least one agent strictly better off. Theorem 2 of Bogomolnaia and Moulin [2004] proves that essentially the same procedure as Algorithm 6 when applied to the random assignment problem with dichotomous preferences yields a group strategyproof mechanism.

Upon closer examination of the proof of Theorem 2, though the proof was written specifically in the context of random assignment, the same ideas essentially carry over to the cake cutting setting. The proof relies on similar ideas, essentially arguing that by deviating it is not possible to gain more of the intervals that are also desired by the non-deviating agents. The proof itself is very specific to the random assignment setting and argues about allocation matrices Z that specify what fraction of each item each agent receives. However, if we instead view Z not as an allocation matrix but instead as a specification of a cake cutting allocation (an assignment of subintervals of $[0, 1]$) to each agent, then the proof carries through for the cake cutting setting as well. As a result, their proof shows that not only is Algorithm 6 truthful, but it also satisfies the stronger notion of group strategyproofness. We do not make this a main part of our Theorem 6.4.1 since it was not an independent contribution of ours.

6.5 Randomized Mechanisms

In the previous section we saw that designing deterministic truthful and fair mechanisms is not an easy task, even if the valuation functions of the agents are rather restricted. In

this section we demonstrate that by allowing randomness we can obtain significantly more general results.

A *randomized cake cutting mechanism* outputs a random allocation given the reported valuation functions of the agents. There are very few previous papers regarding randomized mechanisms for cake cutting. A rare example is the paper by Edmonds and Pruhs [2006b], where they give a randomized mechanism that achieves approximate proportionality with high probability. We are looking for a more stringent notion of fairness. We say that a randomized mechanism is *universally proportional* (resp., *universally EF*) if it always returns an allocation that is proportional (resp., EF).

One could also ask for *universal truthfulness*, that is, require that an agent may never benefit from lying, regardless of the randomness of the mechanism. A universally truthful mechanism is simply a probability distribution over deterministic truthful mechanisms. However, asking for both universal fairness and universal truthfulness would not allow us to enjoy the additional flexibility that randomization provides. Therefore, we slightly relax our truthfulness requirement. Informally, we say that a randomized mechanism is *truthful in expectation* if, for all possible valuation functions of the other agents, the expected value an agent receives for its allocation cannot be increased by lying, where the expectation is taken over the randomness of the mechanism.

We remark that while truthfulness in expectation seems natural, consistent as it is with expected utility maximization, fairness (i.e., proportionality and envy-freeness) is something that we would like to hold *ex-post*; fairness is a property of the specific allocation that is being made, and continues to be relevant after the mechanism has terminated. Interestingly enough, if we were to turn this around, then achieving universal truthfulness and envy-freeness/proportionality in expectation is trivial: simply allocate the entire cake to a uniformly random agent!

There is a simple class of randomized mechanisms which are truthful in expectation. Specifically, define a *partition mechanism* as follows. On input $V_1, \ldots, V_n$ a partition mechanism chooses a partition $\{X_1, \ldots, X_n\}$ of [0,1]. The mechanism then uniformly samples a random permutation π over the n agents, and assigns agent i the piece $X_{\pi(i)}$. If an agent reports truthfully, then in expectation, it receives value $\sum_{j=1}^{n} \frac{1}{n} V_i(X_j) = 1/n$. If the agent reports some other valuation V_i', then the partition the mechanism chooses may change to $\{X_1', \ldots, X_n'\}$, but the agent still receives value $\sum_{j=1}^{n} \frac{1}{n} V_i(X_j') = 1/n$.

However, depending on the partitions chosen, a partition mechanism may not be universally proportional and EF. In order to obtain a mechanism that is universally proportional

and EF, we require that the partitions selected are *perfect partitions*. A partition is perfect for reports $V_1, \ldots, V_n$ if $V_i(X_j) = 1/n$ for every i and j. As the agents have value exactly $1/n$ for every element of the partition, a partition which always selects perfect partitions is clearly universally proportional and EF.[8]

Though partition mechanisms which select perfect partitions are truthful in expectation and universally proportional and EF, there still remains the obstacle of actually finding a perfect partition given the valuation functions of the agents. Does such a partition exist, and can it be computed? More than sixty years ago, Neyman [1946] proved that if the valuation functions of the agents are defined by the integral of a continuous probability measure then there *exists* a perfect partition. Unfortunately, Neyman's proof is nonconstructive, and to this day there is no known constructive method under general assumptions on the valuation functions. This is not surprising since a perfect partition induces an EF allocation, and finding an EF allocation in a bounded number of steps for more than four agents is an open problem.

However, if we assume more structure on agent preferences, then it is possible to explicitly construct perfect partitions. As an example, consider the class of piecewise constant valuation functions. Each agent makes a mark at the left and right boundaries of each of the intervals it is interested in, and we add two marks at 0 and 1. The value density function of each agent is constant over each subinterval between two consecutive marks, as this subinterval is either contained in one of the subintervals agent i is interested in or completely disjoint from any of them. Hence, the allocation that assigns to each agent exactly $1/n$ of each of the subintervals between two consecutive marks is a perfect partition. A similar procedure can be applied to piecewise linear valuation functions by observing that if agent densities are linear on a subinterval, we can partition the subinterval into n elements that have the same value to each agent. Specifically, we can divide the subinterval into $2n$ pieces of equal length and label the pieces $1, \ldots, 2n$ from left to right. To construct element i of the partition, we match piece i with piece $2n - i + 1$. An agent has the same value for every element of the partition due to the assumption of linear densities on the subinterval.

[8] Mossel and Tamuz [2010] also observe that partition mechanisms which select perfect partitions are truthful in expectation and universally proportional.

6.6 Discussion

The free disposal assumption

For our deterministic result we rely heavily on the ability to throw undesired pieces of the cake away. The reason this is important is that it prevents agents from being able to gain by reporting smaller intervals, in hopes of gaining a larger share of the overlap while obtaining desired intervals for free. The following example illustrates why removing the free disposal assumption can be problematic.

Since we can no longer discard undesired intervals, we must specify how to allocate the undesired intervals. Suppose we decide to allocate undesired intervals evenly between agents 1 and 2, giving the left half to agent 1. While this seems benign, our mechanism is no longer truthful.

Suppose there are two agents. Consider the allocation made when both agents report that they desire the interval $[0, 0.2]$. Mechanism 6 calls for any allocation that splits $[0, 0.2]$ equally between the agents.[9] Assume a simple implementation that gives $[0, 0.1]$ to agent 1 and $[0.1, 0.2]$ to agent 2. Our rule for allocating undesired intervals gives $[0.2, 0.6]$ to agent 1 and $[0.6, 1]$ to agent 2.

Now consider the case where agent 1 values $[0, 0.6]$ and agent 2 values $[0, 0.2]$. If the agents report truthfully, Mechanism 6 gives $[0, 0.2]$ to agent 2 and $[0.2, 0.6]$ to agent 1. Agent 1 receives $[0.6, 0.8]$ and agent 2 receives $[0.8, 1]$ due to our rule for allocating undesired intervals. Agent 1 receives value $2/3$. On the other hand, if agent 1 reports $[0, 0.2]$ (assuming agent 2 remains truthful), it receives $[0, 0.1] \cup [0.2, 0.6]$ which gives value $5/6$. Notice that the problem here persists even if we allocate $[0.2, 0.6]$ to agent 2 when both agents report $[0, 0.2]$. In this case, agent 2 can deviate when it has true value $[0, 0.6]$ and agent 1 reports $[0, 0.2]$.

It is unclear whether we can dispose of the free disposal assumption. Indeed, it may be that there is an impossibility result when we force all intervals to be allocated. It would be nice to be able to apply the same insights of Mechanism 6, though as the example shows, we now have to be careful about the exact mechanics of *how* intervals are allocated rather than simply the fraction of each interval each agent receives. This adds significant complexity to the problem which we are able to avoid in Mechanism 6 when we assume free disposal.

[9] In Mechanism 6, any exact allocation suffices in FairAllocate, and it is not necessary to worry about the specific way an exact allocation is achieved.

6.7 Summary and Future Work

Departing from the discussion in Chapters 4 and 5, which do not consider strategic issues, this chapter considers the question of finding DSIC mechanisms for cake cutting under restricted valuations. The main result is a polynomial time DSIC mechanism that is proportional, EF, and PE when agents have piecewise uniform valuations. A secondary result shows that if randomized mechanisms are allowed, there exists a polynomial time randomized mechanism that always finds an allocation that is proportional and EF when agents have piecewise linear valuations.

6.7.1 Future Work

1. The most prominent technical challenge is to generalize Mechanism 6. The first step would be a deterministic, truthful, proportional, and EF mechanism under the assumption that the agents have piecewise constant valuations, and the second step would be achieving the same result with respect to piecewise linear valuations. While our mechanism for piecewise uniform valuations is Pareto-efficient, Schummer [1997] proves that in a setting of pre-defined divisible goods, the only Pareto-efficient and truthful mechanism is a dictatorship that gives all goods to a single agent. Since piecewise constant valuations contain this setting (by creating separate intervals for each good and letting agents express their level of utility for each good by varying their utility function on each interval), we cannot hope for truthfulness, Pareto-efficiency, and fairness when we move to piecewise constant valuations. However, we might still be able to attain truthfulness and fairness (proportionality and envy-freeness). Unfortunately we cannot rule out the scenario where no such mechanisms exist; future work would have to resolve this issue.

2. As we have done throughout this thesis, we have worked with restricted valuations and a direct revelation model.

   It is unclear whether our mechanisms can be efficiently implemented via evaluation and cut queries. In particular, it is unclear whether it is possible to exactly isolate the agents' desired intervals using a finite number of queries. This does not preclude, however, the existence of results that are analogous to ours via mechanisms that only rely on evaluation and cut queries.

Appendix: Equivalence of Swapping Procedure and Mechanism 6

We prove that the swapping procedure from Section 6.4.2 is equivalent to Mechanism 6 for two agents. In the swapping procedure from Section 6.4.2, there are two cases. The first is where agent 2 gives away all of its share of W_{12}, and the second is where agents 1 and 2 divide W_{12}. We can give precise conditions for when each of the two cases occurs. After the first set of swaps, agent 1 has all of W_1, and agent 2 has $\text{len}(W_2 \cup W_1)/2$ of W_2 and still desires the $(\text{len}(W_2) - \text{len}(W_1))/2$ of W_2 still held by agent 1. If $\text{len}(W_{12})/2 \geq (\text{len}(W_2) - \text{len}(W_1))/2$ then agent 2 can get all of W_2 without exhausting all of its share of W_{12}, and the agents equally split $W_1 \cup W_2 \cup W_{12}$. Otherwise, agent 1 receives all of W_1, W_{12} and agent 2 receives only W_2.

The crux of the swapping procedure is the allocation of W_{12}. Each agent always receives W_i, and W_{12} is allocated to try and equalize the total lengths of intervals obtained by each agent. However, this may not always be possible because agent 1 may desire less than $\text{len}(W_1 \cup W_2 \cup W_{12})/2$, and in this case, agent 1 receives all of W_{12}. To summarize, there are two cases in the swapping procedure:

1. $\text{len}(W_1 \cup W_{12}) \geq \text{len}(W_1 \cup W_2 \cup W_{12})/2$. Agent 1 receives W_1, agent 2 receives W_2 and the agents split W_{12} so that their allocated lengths are equal.

2. $\text{len}(W_1 \cup W_{12}) < \text{len}(W_1 \cup W_2 \cup W_{12})/2$. Agent 1 receives $W_1 \cup W_{12}$ and agent 2 receives W_2.

To see that this swapping procedure is exactly equivalent to Mechanism 6 for two agents, assume again without loss of generality that $\text{len}(W_1) \leq \text{len}(W_2)$. In Mechanism 6, either $\{1\}$ or $\{1, 2\}$ is the subset with the smallest average. $\{1, 2\}$ is the chosen subset if $\text{len}(W_1) + \text{len}(W_{12}) \geq (\text{len}(W_1 \cup W_2 \cup W_{12})/2$. If $\{1, 2\}$ is chosen, then agents 1 and 2 split $W_1 \cup W_2 \cup W_{12}$ in an exact allocation. If $\{1\}$ is chosen, then agent 1 receives all of W_1, W_{12}, and agent 2 receives all of W_2.

Chapter 7

Combinatorial Auctions

Combinatorial auctions are a canonical problem at the heart of the intersection of computer science and economics. In combinatorial auctions, multiple items are auctioned at the same time. We assume that agent valuations are private and must be elicited by the auction mechanism. On the economic side, these auctions are desirable as they naturally accommodate complex bidder preferences for the goods.

As an example, suppose that two items exhibit strong complementarities. More concretely, consider an auction for wireless spectrum, where spectrum rights for different regions of the US are being auctioned. Verizon gains a significant bonus if it is able to obtain rights in all the regions and build a national network. A combinatorial auction allows this to be expressed as all the regions are allocated in a single auction.

On the other hand, running separate auctions for each region would force Verizon to guess about its probability of obtaining subsequent regions when bidding for a region. In particular, in hopes of obtaining a national network, Verizon is willing to pay more than its value for regional spectrum. If Verizon ultimately does not receive a national network (for instance because of an aggresive regional provider), it could be left paying more than its value for the spectrum it actually receives. This is known as the *exposure* problem and is one of the arguments in favor of combinatorial auctions. Indeed, combinatorial auctions have been adopted to allocate wireless spectrum in many different countries (see e.g. [Cramton, 2002]), to find bus route operators in London [Cantillon and Pesendorfer, 2006], to procure goods and services in industrial settings [Caplice and Sheffi, 2006, Bichler et al., 2006, Sandholm, 2013].

Though combinatorial auctions are able to accommodate complex bidder preferences, allocating a set of items altogether gives rise to interesting computational challenges. If

we have m items, then there are 2^m possible subsets of items that an agent can obtain. A fully general preference would allow for the bidder to specify a different value for every possible subset of items. Even if we impose natural restrictions such as monotonicity (i.e., an agent's value for a superset of a set S is at least its value for S), this still does not decrease the number of values needed to pin down a bidder's valuation. As a result, combinatorial auctions give rise to a *preference elicitation* problem where we need to learn enough about an agent's preference so we can make an intelligent allocation decisions without forcing the agent to communicate its entire valuation function since that is computationally intractable (in a worst case sense). We will not address preference elicitation in this dissertation, but more information on the topic can be found in Cramton et al. [2006]. Instead, we limit the valuations we consider to families of valuations that can be succinctly communicated (e.g. by listing values for target bundles of interest and having rules for how these translate to values for any bundle of items).

Another computational issue that arises is the winner determination problem. Suppose that we are given agent preferences and wish to find an allocation that maximizes social welfare. It turns out that in many combinatorial auction settings this problem is computationally intractable. In this chapter, we present the model for combinatorial auctions as well as some basic results that will be relevant to the next two chapters on monotone branch and bound search and learning payment rules.

7.1 Model

In the combinatorial auction (CA) problem, there is a set N of agents and set G of items, with $|N| = n$, $|G| = m$. Each agent has a private *valuation function* $v_i : 2^G \to \mathbb{R}_{\geq 0}$ which expresses the agent's value for each possible bundle of items. We adopt the same notation as Section 2.2.[10] A *valuation profile* $(v_1, \ldots, v_n) = v$ consists of a valuation function for each agent. It will be useful to write a valuation profile from the perspective of agent i as $v = (v_i, v_{-i})$, where v_i gives agent i's valuation function and v_{-i} refers to the valuation function of all other agents.

The combinatorial auction problem is a mechanism design with money problem (Section 2.2), and we make the typical assumption that agent utilities are quasi-linear. The set of possible outcomes Ω is an assignment of items to each agent such that no item is given to

[10]The notation in Section 2.2 uses a capital V_i for agent valuations, and this is used in our discussion of cake cutting as well. In our discussion of CAs and subsequent chapters, we use a lowercase v_i for agent valuations.

multiple agents. We call elements of Ω *allocations*. The mechanism design goal is to find an outcome rule g that maps a valuation profile v to an element of Ω and a payment rule that maps a valuation profile v to a payment for each agent such that g chooses an element of Ω that has good properties for each v, and g and p together incentivize agents to report their valuations truthfully.

7.2 Computational Complexity

Even if we momentarily ignore the incentive problem of designing g and p so that agents report their valuations truthfully, a fundamental computational problem arises when we consider CAs and social welfare maximization. Suppose that our objective as a designer is to find an allocation that maximizes social welfare with respect to the reported valuations. This problem is known as the *winner determination problem*. This problem is related to a weighted set-packing problem and is NP-hard, even if we limit ourselves to fairly restrictive preferences. [Cramton et al., 2006] contains a full discussion, and we summarize the parts relevant to this thesis in this section.

Before we discuss computational complexity, we must define precisely the inputs to our algorithm. One possibility is to say that the inputs are an exhaustive specification of the agents' valuation functions. That is, for each agent, the input consists of 2^m numbers, with each number representing the agent's value for some subset of items. One problem with this approach is that even communicating these numbers becomes infeasible as m grows large. Another problem is that if we use this representation, then a polynomial time algorithm would allow time that scales polynomially with n and 2^m rather than polynomially in n and m. As a result, we need to consider more succinct input representations or *bidding languages*.

Two fundamental bidding languages are the OR and XOR bidding languages [Sandholm, 2002, Fujishima et al., 1999]. In both languages, agents submit lists of pairs $(S_j, v_i(S_j))$ where $S_j \subseteq G$. The OR language interprets the bids as being additive as long as the subsets listed do not intersect. If an agent submits $\{(S_1, v_i(S_1)), (S_2, v_i(S_2))\}$ with $S_1 \cap S_2 = \emptyset$, then it is assumed that the agent receives value $v_i(S_1) + v_i(S_2)$ for the subset $S_1 \cup S_2$.[11] On the other hand, the XOR language assumes that bids are mutually exclusive. That is, at most one of the bids can be accepted, so in the given example, if the agent were given $S_1 \cup S_2$, its value would be $\max(v_i(S_1), v_i(S_2))$. It is also possible to combine the two languages

[11] In general if we have overlapping bids then an agent's value for S is the maximum sum of values for disjoint bids contained in S.

by creating dummy items, and this is known as the OR* language [Fujishima et al., 1999]. The properties of these languages is beyond the scope of the thesis, but interested reader is referred to [Nisan, 2006].

With OR, XOR, or OR* bids, we now have a natural input size for winner determination. Because agents list specific bundles and values in these bidding languages, we can use the number of specific bundles listed as the input size. We then seek algorithms that run in time polynomial in the number of (bundle, value) pairs submitted by the agents as well as n and m. We now specialize to the XOR bidding language, but similar intractability results hold for the OR and OR* languages [Sandholm, 2006]. To discuss computational complexity, we consider the decision version of winner determination, which asks, for a given set of XOR bids, whether there exists a feasible allocation that has value at most K, where K is any positive number.

Theorem 7.2.1. *[Rothkopf et al., 1998] The decision problem for winner determination with XOR bids is NP-hard.*

The proof shows that winner determination is equivalent to a weighted set packing problem. This negative result holds even if agents are *single-minded* and can only submit a single (bid, value) pair. Additionally, even when agents are single-minded, it is not possible to approximate the optimal allocation in polynomial time.

Theorem 7.2.2. *[Sandholm, 2002, Lehmann et al., 2002] It is not possible for polynomial time algorithms to guarantee a solution with social welfare at least $m^{-1/2+\epsilon}$ times the optimal social welfare. The result requires that $NP \neq ZPP$, where ZPP is the class of problems that can be solved using randomized algorithms that always return the correct answer (regardless of the realization of the random coin tosses).*

7.3 Computational Mechanism Design

The negative computational complexity results serve as one motivation for the study of *computational mechanism design* (CMD). CMD imposes that in addition to g and p satisfying incentive constraints, we would like the computation of g and p to be tractable, i.e. take polynomial time in the natural input parameters to the problem.

If we are looking to maximize social welfare in combinatorial auctions and are not considering computational limits, then we can apply the VCG mechanism discussed in Section 2.2.3. However, the VCG mechanism requires us to find the allocation that exactly

optimizes social welfare, and this problem is NP-hard as discussed in the previous section. It is tempting to simply replace g with some algorithm g' that approximately maximizes social welfare and to use a payment rule that shares the same spirit as the VCG payment rule in charging agent i the externality it imposes on the agents. Rather than charging agent i

$$\sum_{j \neq i} v_j(g(v)) - \max_{\omega \in \Omega} \left(\sum_{j \neq i} v_j(\omega) \right),$$

we can charge agent i

$$\sum_{j \neq i} v_j(g'(v)) - \sum_{j \neq i} v_j(g'(v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)).$$

Here we assume that g' is well-defined when passed $n-1$ agents as the right hand side of the expression requires us to compute the allocation when agent i is removed. This is a reasonable assumption for most algorithms that attempt to maximize welfare.

Unfortunately, using g' along with the VCG-style payment rule is not DSIC. One easy way to see why this might be true is to observe that g' might not satisfy the necessary properties for there to exist a payment rule that makes it incentive compatible. For instance, if we are in a restricted CA setting that is single-parameter (such as the known single-minded setting discussed below), our approximation algorithm g' might not be monotone in this single-parameter. As discussed in Section 2.2.3, monotonicity is necessary for there to exist a payment rule that is DSIC when combined with g'. Even if g' satisfies the necessary conditions for there to exist a DSIC payment rule, the VCG-style payment rule may not be the right payment rule.

As a result, it is difficult to rely on classic results like the VCG mechanism if we consider computational constraints in CAs and welfare maximization. We cannot exactly optimize social welfare and we cannot use a VCG-style rule for algorithms that approximately optimize welfare. The challenge is to simultaneously find (a) approximation algorithms that do a good job of maximizing welfare and (b) payment rules that work together with the approximation algorithm to satisfy incentive constraints.

7.4 Single-Minded CAs

There is much recent work in CMD, and it is beyond the scope of this dissertation to broadly survey the state of the field. Section II of [Nisan et al., 2007] provides a more thorough

exposition. Here we introduce the special case of single-minded CAs as they will be relevant for the next two chapters. We also point the reader to Section 1.2.3 and the related work sections of Chapters 8 and 9 for additional background on related work in CMD.

In single-minded CA, each agent is interested in a single bundle. Therefore, an agent's valuation is summarized by its target bundle S_i along with its value for this bundle $v_i(S_i)$. If the target bundle S_i is publicly known, then this becomes a single-parameter setting since an agent's valuation is pinned down by its value for the publicly known target bundle.

While restrictive, there are settings where bidders may be known single-minded. Lehmann et al. [2002] describe a pollution rights auction where companies are bidding for the right to emit certain chemicals into the air, and the pollution profiles of the companies are known. They also describe communication network settings where bidders own nodes in the network and wish to connect their nodes. If there is only a single-path between any pair of nodes, then bidders are single-minded. If it is also public knowledge which companies own which pairs of nodes, then this becomes a known single-minded setting.

While the known single-minded setting is easier than the single-minded setting from an incentives perspective (since an agent can only lie about its value for the target bundle and not the identity of the bundle), computing the welfare-maximizing allocation is still NP-hard as discussed in Section 7.2.

Single-minded CA offer an example of a setting where algorithms that approximately maximize welfare have been used successfully in the context of mechanism design. Lehmann et al. [2002] provide a greedy algorithm and associated payment rule with a $1/\sqrt{m}$ welfare guarantee (relative to the optimal welfare), where m is the number of items being allocated, and a matching lower-bound. More recently, Mu'alem and Nisan [2008] provide an approximation for the special case of known single-minded CAs with guarantee $1/(\epsilon\sqrt{m})$ for any fixed $\epsilon > 0$, with runtime that is exponential in $1/\epsilon^2$. These advancements both provide worst-case guarantees in terms of welfare compared with the optimal and polynomial time computation. The drawback is that in order to obtain these worst-case guarantees and incentive-compatibility it is necessary to be able to reason analytically about the algorithms.

In the next two chapters, we explore a more computational approach to designing mechanisms. In Chapter 8, we consider using branch and bound search as our algorithm for finding an allocation with good welfare. In Chapter 9, we relax our requirement for exact incentive-compatibility and provide a general framework that learns payment rules to pair with any provided outcome rule.

Chapter 8

Monotone Branch and Bound Search

In this chapter, we consider the mechanism design problem for known single-minded CAs. While we focus on this specific setting, our high-level ideas apply to the more general setting of single-parameter downward-closed domains (allocation problems where removing an agent's allocation from a feasible allocation remains feasible). As discussed in Section 7.4, known single-minded CAs are an interesting problem in CMD as the winner determination problem remains NP-hard, despite the severe restrictions on agent valuations and the assumption that agents' target bundles are publicly known. The existing approaches of Lehmann et al. [2002] and Mu'alem and Nisan [2008] give polynomial time algorithms with worst-case guarantees on the welfare of the computed allocation along with appropriate payments that make the resulting mechanism DSIC.

However, if incentives were not a concern, we have more sophisticated algorithms such as Branch-and-Bound (BnB) search, which can efficiently find optimal solutions to the winner determination problem on typical instances. Following a research agenda on *heuristic mechanism design* [Parkes, 2009], we seek to leverage heuristic algorithms such as BnB search for the purpose of CMD.

BnB search is a canonical method for solving optimization problems that are formulated as integer programs (IPs). Search proceeds by branching on decisions in regard to whether or not an agent is allocated ("branch"), and looking to prune large parts of the search space through linear program (LP) relaxations ("bound"). In cases where it is too computationally expensive to compute the optimal solution, an *optimality tolerance* $\gamma \in (0, 1]$ is adopted, and search is terminated when a solution is identified that is proven to be within multiplicative

fraction γ of the optimal solution.

But therein lies the core problem in combining BnB with incentive compatible mechanism design:

(i) a natural adaptation of the payments of the VCG mechanism need not be truthful when coupled with an approximate solution to a welfare optimization problem (see Section 7.3, and

(ii) the allocation generated by BnB search when used with an optimality tolerance $\gamma < 1$ need not be *monotone*, in the sense that an agent might go from winning at some bid value w_i to losing at some bid $w_i' > w_i$.

BnB search is monotone for $\gamma = 1$ because it computes the optimal allocation. But monotonicity can fail with BnB when $\gamma < 1$, because a higher agent value can trigger a different search decision somewhere in the search tree, eventually leading to the search terminating with an alternate solution that is within a factor γ of optimal but does not include the agent.

8.1 Our Results

Correcting this failure of monotonicity, we follow an approach introduced by Parkes and Duong [2007] in a different context. Given an instance, we check to see whether agent i allocated at bid w_i becomes unallocated for any bid $w_i' > w_i$ (fixing the other bids.) If this occurs, then the outcome is "corrected" (or *ironed*) such that the agent is not allocated at bid w_i. By doing this for all inputs, we achieve *monotone BnB search* (and thus incentive-compatibility). Moreover, the approach retains good welfare if the original search algorithm is monotone for most agents on most inputs.

The technical challenge is to find an efficient method to trace the effect on the outcome of BnB search as the bid value of an agent is increased, taking each agent in turn. From the perspective of an IP, we are increasing an objective coefficient and tracing the effect on decisions made during BnB search (e.g., branch decisions and pruning decisions.) The technical innovations involved in making this sensitivity analysis of BnB search efficient include:

• An efficient technique to identify the next highest objective value coefficient at which a different search decision would be made for a given BnB search state.

• Caching search states to avoid re-running early steps of BnB search that remain the same when testing higher objective value coefficients.

• Leveraging structure of BnB search to identify sufficient conditions that ensure that agent i is allocated in any BnB solution, and terminating sensitivity checks early when this is the case.

• Caching of LP solutions to avoid expensive re-computations when the solutions would not have changed.

• Making BnB search more monotone by adopting a bucketing approach to fractional variables in deciding which variable to branch on, and through a discrete transformation on the inputs.

We implement our technique and report experimental results based on the well-studied "legacy" distributions. We did not use the named CATS distributions (matching, paths, regions, scheduling) as there is no straight-forward way to adapt these to the single-minded setting. In particular, we focus on the L4 (decay) distribution, which has been shown in the literature to generate hard winner determination problems Leyton-Brown et al. [2000], Sandholm [2002], Sandholm et al. [2005]. We find sets of randomly generated instances from the L4 distribution where the best parameterizations of our monotone BnB algorithm yield better welfare than the approximation mechanisms of Lehmann et al. [2002] and Mu'alem and Nisan [2008].

Additionally, the best parameterizations of monotone BnB (and for an optimality tolerance $\gamma < 1$ at which welfare is better than existing approximation mechanisms) have better runtime than optimal BnB. Monotone BnB is also fully parallelizable in the number of allocated agents while the same is not true of optimal BnB. The fully parallelized runtime cost of monotone BnB is significantly smaller than that of optimal BnB for the best parameterizations of monotone BnB and instances we consider. Though our experimental results depend crucially on these input distributions, we believe they demonstrate the potential of the general approach and the specific application to BnB search.

In addition, while earlier work has developed techniques for the sensitivity analysis of optimal solutions to IPs [Marsten and Morin, 1977, Feautrier, 1988], we are not aware of earlier work on the sensitivity of BnB search when used with an optimality tolerance. For this reason, we also provide some analysis of the kinds of decisions that tend to change during search and the kinds of monotonicity failures that we see on our instances. We see that on our test instances, the most common changes result from a pivot to a new LP solution, which causes a change in the branch variable selected.

8.2 Related Work

We follow earlier work of Parkes and Duong [2007] and Constantin and Parkes [2009], who have applied so-called "computational ironing" to online stochastic combinatorial optimization (OSCO). BnB search is more complex algorithmically than the OSCO algorithms studied in this earlier work, and requires new technical contributions in finding an efficient coupling with the approach of computational ironing.

Also thematically related to heuristic mechanism design is the GrowRange method of Parkes and Schoenebeck [2004], which provides an anytime algorithm for welfare optimization in general CAs by expanding the range of a VCG-based algorithm, while allowing for a time-based interruption by the center (although without providing full strategyproofness.)

As discussed in Section 1.2.3, there are several papers that look at converting algorithms into incentive compatible algorithms while preserving the welfare performance of the original algorithm [Briest et al., 2005, Lavi and Swamy, 2011, Dughmi and Roughgarden, 2010, Hartline and Lucier, 2010, Hartline et al., 2011]. A key difference with most of this work is that our mechanism is DSIC while most of these previous papers provide randomized truthful in expectation mechanisms or BIC mechanisms. Of these papers, only Briest et al. [2005] examine DSIC. They provide a DSIC mechanism whose outcome rule is an FPTAS for social welfare, assuming the existence of an optimal pseudopolynomial time algorithm for welfare maximization. Single-minded CAs, which we study, do not have an optimal pseudopolynomial time algorithm for welfare maximization, so their main result does not apply to our setting. They do leverage their technical insights to give a new algorithm for multi-unit CAs where there are at least B copies of each item. However, for the single-minded CA problem we study B may equal 1 (we may have only one copy of each item), and their new algorithm does yield optimal worst-case approximation ratios (in contrast to the algorithms of Lehmann et al. [2002] and Mu'alem and Nisan [2008]). In addition to the specific differences with respect to the work of Briest et al. [2005], we are not aware of any computational validation of these other approaches. Indeed, one of our main contributions is to implement and experimentally test our approach on distributions studied in the literature.

8.3 Preliminaries

In known single-minded CAs each agent has a target bundle T_i, known to the mechanism, and a value $w_i > 0$ for this bundle. We thus refer to agent reports as being this single value w_i, rather than a report of an agent's entire valuation function v_i. Also, we consider

deterministic allocation functions, so we can assume that $g_i(w_i, w_{-i}) \in \{0, 1\}$ corresponding to whether or not the agent is allocated its target bundle.

Definition 8.3.1. An allocation function g is monotone in a known single minded-domain if $g_i(w_i, w_{-i}) = 1 \Rightarrow g_i(w_i', w_{-i}) = 1$ for all $w_i' \geq w_i$.

As discussed in Section 2.2.3, for deterministic allocation functions g and single-parameter agent preferences, there exists a payment function p that makes (g, p) truthful iff g is monotone. In fact, once the allocation function is known, the payment function can be computed by finding the "critical value" at which an agent starts receiving its target bundle. As a result, for single-dimensional settings such as known single-minded CAs, the problem of constructing truthful mechanisms can be reduced to that of finding monotone allocation functions. In the context of the proposed framework, once an allocation has been confirmed for an agent by performing a check of monotonicity for all higher reports the agent could have made, then a parallel, *downward* sensitivity check is performed to find the *first* smaller value at which the agent would no longer be allocated.

Known single-minded CAs are a special case of the more general class of downward closed environments.

Definition 8.3.2. A single-dimensional mechanism design environment is *downward closed* if an outcome is described by a set of agents allocated, and for a given outcome, there exists an outcome that is associated with any subset of the allocated agents.

In the known single-minded setting, wlog, we can specify an outcome by giving the set of agents that receive their target bundle. Technically there are many feasible allocations that are affiliated with satisfying a given set of agents, but we can limit consideration to allocations that do not give agents more than their target bundle since this agents get no additional value from additional items in the single-minded setting. The downward closed property holds since if it is possible to give a set of agents each their target bundle, then we can satisfy any subset of that set as well. While we focus on known single-minded CAs, the general ironing procedure developed in Section 8.4 applies to single-dimensional, downward closed environments.

8.4 Ironing, Discretization and a First Approach

We first describe the very basic approach to making heuristic algorithms monotone for single-parameter, downward closed domains. In the next sections, we propose techniques to reduce the computational overhead in the particular context of BnB search.

The basic idea of ironing is straightforward. We first compute the set of allocated agents using our allocation algorithm at the current values. We then perform sensitivity analysis on the set of allocated agents. For each allocated agent, we check if the agent would still be allocated under the allocation algorithm for all higher reported values. If an agent becomes deallocated for higher reported values, then we must deallocate the agent since this indicates a non-monotonicity in the provided allocation function. This general procedure is described as *ironed-alloc* in Algorithm 7. We focus on the allocation function in the body of the paper, but the same ideas can be applied to compute payments for allocated agents by performing downward sensitivity rather than upward sensitivity.

Theorem 8.4.1. *The* ironed-alloc *procedure is monotone and feasible for downward closed domains.*

Proof. An agent is allocated in *ironed-alloc* only if the agent is allocated at its current value and all higher reported values. If this is the case, then *ironed-alloc* would still have allocated the agent for all higher reports. □

Algorithm 7 *ironed-alloc*(*alloc-func*, values)

```
allocated = alloc-func(values)
for agent ∈ allocated do
  if is-deallocated-at-higher-values(agent) then
    allocated = allocated \ agent
  end if
end for
return allocated
```

Algorithm 8 *discretized-ironed-alloc*(*alloc-func*, values, β)

```
for value ∈ values do
  value = ⌊value/β⌋β
end for
allocated = alloc-func(values)
for agent ∈ allocated do
  if is-deallocated-at-higher-values(agent) then
    allocated = allocated \ agent
  end if
end for
return allocated
```

If the underlying allocation function is monotone everywhere, then *ironed-alloc* will be the same as the underlying algorithm. If it is not, then *ironed-alloc* may sacrifice welfare (since it must deallocate some agents) in order to preserve monotonicity.

As stated, *ironed-alloc* applies to continuous type domains as long as we have a method for *sensitivity checking*; i.e., to determine whether an agent will become deallocated for any higher reports. However, such a procedure may not always be available, and even when it is, implementing such procedures in practice may introduce an implicit discretization.[12]

For this reason, we introduce a discretized version of *ironed-alloc* that is monotone in the original domain, even if the original domain is continuous, and still results in payments that are individually rational. The procedure mimics *ironed-alloc*, except that agent bids are rounded down to the nearest grid size β (Figure 7).

Theorem 8.4.2. *Procedure* discretized-ironed-alloc *is monotone and admits individually rational payments.*

Proof. The proof of monotonicity is the same as Theorem 8.4.1. To see that payments are individually rational, recall that given a monotone allocation function, the payment of an agent is the lowest value at which the agent would still be allocated. Suppose that an agent is allocated when bidding w_i. This means that the agent would also be allocated with bid $\lfloor w_i/\beta \rfloor \beta \leq w_i$, so the agent's payment is at most w_i. □

With grid size β, we can obtain a procedure *is-deallocated-at-higher-values* by testing all multiples of β that are greater than the agent's current value, to see whether the agent would still be allocated. Because *discretized-ironed-alloc* rounds values down to multiples of β prior to sending them to the allocation function, this will capture all possible points where the agent could have become deallocated. We refer to this as the *brute force* sensitivity method.

There is an interesting trade-off in using discretization in the context of ironing. On one hand, the allocation function no longer accesses exact agent values, which can result in allocations with lower welfare compared to the allocations computed using the true values. On the other hand, adopting a discretization may actually improve the "ironed" welfare because there are fewer points where the algorithm is required to still allocate the agent, and as a result, the underlying algorithm may become more monotone and deallocate fewer agents.

[12] Initially, we developed our sensitivity checking procedure for continuous values. We identified sensitivity points w_1 based on whether the search might change for any value strictly greater than w_1 (open) or for any value weakly greater than w_1 (closed). To handle open points, we needed to introduce a parameter ϵ to jump the agent's value to $w_1 + \epsilon$ when running counter-factuals. This discussion will be clearer after Section 8.6.

8.5 Branch-and-Bound Search for Single-Minded CAs

An empirically effective way to find an allocation with good welfare for CAs is to formulate the problem as an integer program (IP) and use BnB search. We describe the essentials of this approach in this section.

We assume that we are in a known single-minded setting with n agents and m items and that each agent is interested in a single bundle T_i and reports value w_i to the mechanism. We can write the following winner determination IP (WDIP) to solve for the welfare-maximizing allocation:

$$\text{maximize} \sum_{i=1}^{n} w_i x_i \tag{8.1}$$

$$\text{subject to} \quad \sum_{i:j\in T_i} x_i \leq 1,\ 1 \leq j \leq m \tag{8.2}$$

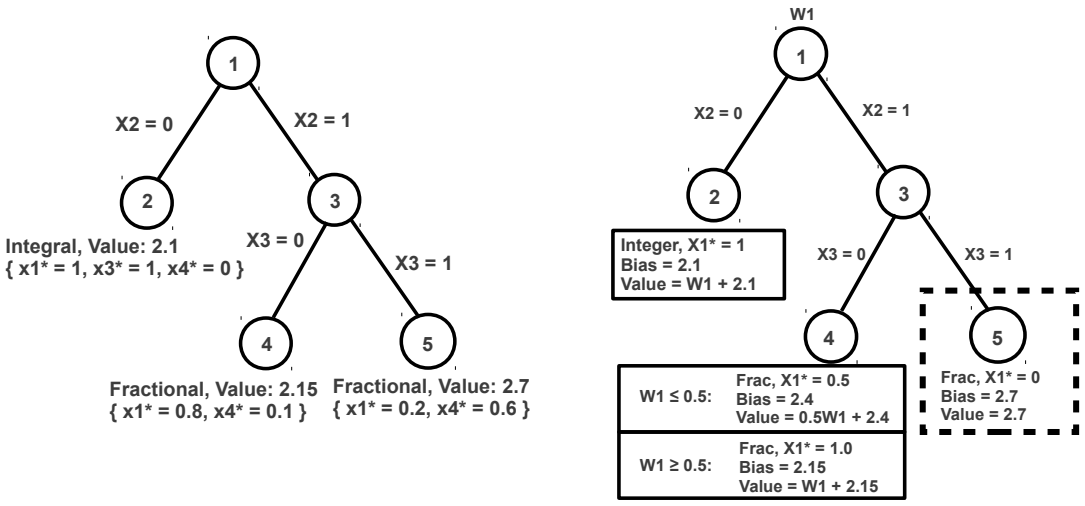
$$x_i \in \{0,1\},\ 1 \leq i \leq n \tag{8.3}$$

The linear programming (LP) relaxation of this IP is the same program, except with the integer constraints (8.3) replaced by inequalities of the form $0 \leq x_i \leq 1$. Given this, Branch-and-Bound (BnB) is a *tree search* technique that uses the relationship between an IP and its LP relaxation to prune parts of the search tree. We will focus on the case where the variables are binary (0 or 1) since the IPs we consider will have this form.

The basic components of the search are the nodes in the search tree. Each node k stores an integer partial assignment, i.e. $t = \{x_2 = 0, x_4 = 1\}$, along with a solution $\{x_1^*, \ldots, x_n^*\}$ to LP_t, where LP_t is the LP relaxation of WDIP, with extra constraints added to enforce t. Let $t(k)$ denote the partial assignment stored in k. With a slight abuse of notation, we say $j \in t(k)$ if x_j is set to 0 or 1 in $t(k)$.

Let the *value of an LP* be $\sum_{i=1}^{n} w_i x_i^*$, where $x_1^*, \ldots, x_n^*$ is the solution to the LP, and the *value of a node* $k = val(k)$ be the value of its LP relaxation. Because the value of an LP relaxation is an upper bound on its associated IP, $val(k)$ is an upper bound on any integer solution that agrees with $t(k)$. A solution $\{x_i^*\}$ is *integral* if $x_i^* \in \{0,1\}\ \forall i$, and *fractional* otherwise. In Figure 8.1(a), node 1 corresponds to an empty partial assignment, while nodes 2 and 3 correspond to partial assignments $\{x_2 = 0\}$, $\{x_2 = 1\}$ respectively. This indicates that x_2 is set to 0 in node 2 and all its children, while x_2 is set to 1 in node 3 and all its children.

A *search tree* has a *root node* with an empty partial assignment, and other nodes are



(a) (b)

Figure 8.1: (a) A simple illustration of a Branch-and-Bound search tree. (b) An illustration of the augmented search state and *get-sens-single-state*.

either an internal node with two children or a leaf node. The left child of an internal node k corresponds to adding $x_j = 0$ to $t(k)$ while the right child corresponds to adding $x_j = 1$ to $t(k)$, for some $j \notin t(k)$. An important property of a search tree is that any integer solution agrees with the partial solution in exactly one leaf of the search tree, i.e. the leaves of any search tree partition the space of possible integer solutions.

The *search state* s is a collection of nodes, and corresponds to the leaf nodes in a valid search tree. $I(s)$ denotes the integral nodes associated with s, $F(s)$ the fractional nodes associated with s, and $K(s) = I(s) \cup F(s)$ all nodes associated with s. In Figure 8.1(a), the search state consists of nodes 2, 4, 5, with node 2 integral and nodes 4 and 5 fractional. Given a search state s, we define the $dec(s)$ to be the *decision* associated with s. To specify the decision, we assume that BnB is being run to an *optimality tolerance* $\gamma \in (0, 1]$, where $\gamma = 1$ represents full optimality. The search decision consists of:

1. Whether or not to terminate the search because a solution with welfare at least γ times the optimal has been found.

2. If the search is terminated, a node $k \in I(s)$ that has the highest value.

3. If the search is not terminated,

   (a) A node $k \in F(s)$ to be selected.

   (b) A variable x_j to be branched.

The crux of BnB lies in how the decision associated with s is computed. We define:

$$UB(s) = \max_{k:k\in F(s)} val(k), LB(s) = \max_{k:k\in I(s)} val(k)$$

If $\gamma \cdot UB(s) \leq LB(s)$, then terminate, and select a node $k \in I(s)$ that has value $LB(s)$. If $\gamma \cdot UB(s) > LB(s)$, then select a node $k \in F(s), \gamma \cdot val(k) > LB(s)$ to be explored (*select-node*) along with a variable x_j to be branched (*branch-variable*). Altogether, the BnB procedure proceeds as:

1. Initialize s to be a single node corresponding to the empty partial assignment.
2. Repeat until termination:
   - Compute $dec(s)$.
   - If terminate, return the integral solution in the node given by $dec(s)$ and terminate.
   - If not terminate, update s by replacing the node given by $dec(s)$ with two children corresponding to branching the variable x_j given by $dec(s)$.

There are various choices for how to implement the *select-node* and *branch-variable* functions. For instance, *select-node* can choose the deepest node, breaking ties by value, (*depth-first*) or choose the node with the highest LP solution value (*breadth-first*) or alternate between the two. A popular choice for *branch-variable* is to select the most fractional variable in the LP solution, but other choices are also possible (see e.g. Chapter II.4 in [Nemhauser and Wolsey, 1998]). The best choices for these functions are typically domain specific. In our work, we choose depth-first for *select-node* until an integral node is found, after which point we use breadth-first. For *branch-variable*, we focus on variants of selecting the most fractional variable.

Upon termination, BnB will return a solution with welfare at least γ times the optimal. This is true because at each step $\max(LB(s), UB(s))$ is an upper bound on the value of any integer solution to WDIP because of the admissibility (or optimistic) estimate of value that comes from the use of LP relaxations and because the nodes in s partition the space of integer solutions.

8.6 Optimized Sensitivity Checking for BnB

In this section, we demonstrate an optimized sensitivity checker (i.e. an implementation of *is-deallocated-at-higher-values*) that takes advantage of the structure of BnB search. In what follows, we assume that we are performing sensitivity checking in the context of *discrete-ironed-alloc*, and can therefore assume that input values are multiples of β. To avoid LP degeneracy (which is problematic for sensitivity because we are unsure which solution will be picked for higher agent values), in our experiments, we add a random value in $[0, \beta)$ to the discretized agent values. This perturbation is independent of an agent's report and thus does not affect truthfulness. In practice, to maintain individual rationality, one would want to subtract a random value, but our implementation adds a random value to avoid special casing perturbations that lead to negative values. This should not have any substantive effect on our experimental results, as properties of the solution are always computed with respect to the original values prior to any discretization or perturbation. For the duration of this section, we assume without loss of generality that we are performing sensitivity checking for agent 1.

Rather than re-run the search for every higher multiple of β, we would ideally like to skip multiples of β that provably continue allocating agent 1 in the solution returned by BnB. The core of such a procedure would consist of a function *get-sensitivity* that runs BnB with agent 1's value set to w_1, but in addition to returning an allocation, returns the next value $w_1' > w_1$ for which we should re-run the search. We could then use the following procedure (summarized in Algorithm 9) as a replacement for the brute force sensitivity checker. We first run *get-sensitivity* with agent 1's reported value. This returns the allocation BnB would have returned, along with the next higher value w_1' at which the allocation might change. We set agent 1's value to w_1', and re-run *get-sensitivity*. If the allocation returned continues to allocate agent 1, then continue the process. Terminate if an allocation returned does not allocate agent 1 or if the next highest value exceeds the maximum allowed value. If the possible values are uncapped, we could always set a very high max value, and treat (the rare case of) any reports greater than this value as being the max value.

The next two sections are devoted to defining *get-sensitivity*. We first examine how a change in w_1 affects a specific node in the search state, and we then use these observations to provide an implementation for *get-sensitivity*.

Algorithm 9 *optimized-is-deallocated-at-higher-values*(*alloc-func*, values, agent, β)

```
sens-value = values(agent)
while sens-value < max-value do
  values(agent) = sens-value
  alloc, next value = get-sensitivity(alloc-func,values, agent, β)
  if agent ∉ alloc then
    return true
  end if
  sens-value = next-value
end while
return false
```

8.6.1 Impact of a Change in Value on a Search Node

We first examine how a *node* in the search state (recall, associated with an LP) changes when agent 1's reported value increases. We separate these changes into two types.

Solution value changes

As a agent 1's reported value increases, the solution to the LP relaxation in a given node may not change, but the value of the solution will change if $x_1^* > 0$ in the solution. If we assume that the solution does not change, then we can easily track how the solution value changes as the agent's reported value increases. Let $x_1^*, \ldots, x_n^*$ be the fractional solution to the LP relaxation at a node. The solution value as a function of agent 1's report w_1 is

$$val(w_1) = w_1 x_1^* + \left(\sum_{i=2}^{n} w_i x_i^* \right),$$

where the expression within the parenthesis does not depend on w_1. We call x_1^* the *coefficient* and the term in parenthesis the *bias* of the node.

LP solution changes

As an agent's reported value increases, the solution to the LP relaxation at the node can change. LPs have the property that solutions lie at corners of the polyhedron formed by the constraints of the LP. The solution stays the same for a range of values, until the agent's value reaches a critical point where the LP is degenerate, and two solutions share the same value. Above this, the new solution becomes the unique optimal solution. The literature on LPs provides simple computational procedures for computing the sensitivity of an LP solution to coefficients in the objective function (see e.g. Section 5.1, [Bertsimas

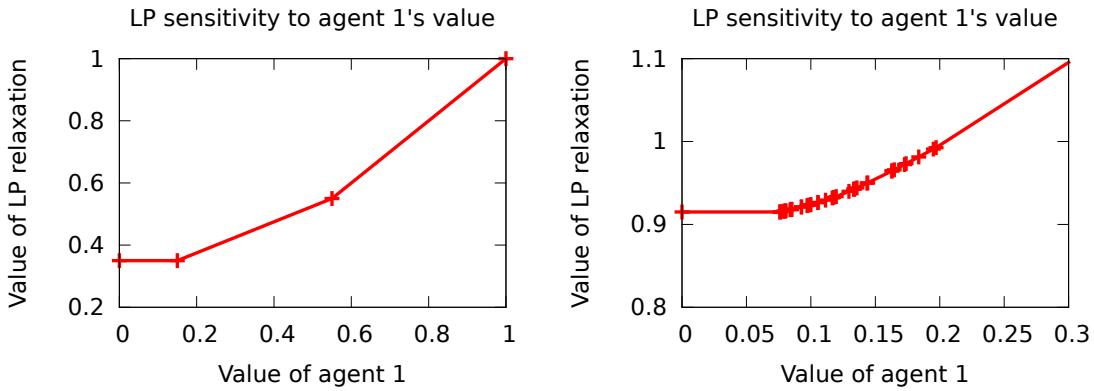


Figure 8.2: Sensitivity of the LP solution value to changes in agent 1's value.

and Tsitsiklis, 1997]).

Worked example

Figure 8.2 illustrates these two types of changes by charting how the value of the LP relaxation changes as the value of agent 1 moves from 0 to 1. The figure on the left represents the simple case where the LP corresponds to the root node of the following instance with 3 agents and 5 goods: Agent 1 desires $\{A, D, E\}$, agent 2 desires $\{A, B\}$ at 0.2, and agent 3 desires $\{B, E\}$ at 0.35. When agent 1's value is 0, the LP solution sets $x_2^* = 0, x_3^* = 1$ and has value 0.35. When agent 1's value reaches 0.15, the LP solution changes to $x_1^* = 0.5, x_2^* = 0.5, x_3^* = 0.5$. At this point, the first and second solutions have the same value. When agent 1's value reaches 0.55, agent 1 becomes fully allocated with $x_1^* = 1$. In $[0, 0.15]$, the value of the LP solution does not depend on agent 1's value. In $[0.15, 0.55]$ the value of the LP solution has slope 0.5, and in $[0.55, 0.1]$, the value of the LP solution has slope 1.0. The figure on the right depicts a more complicated example for the root node LP of an instance with 300 agents. Each marked point on the graph depicts a point where the LP solution changes, and within marked points, the value of the LP solution is linear in the value of agent 1, with the slope governed by the assignment x_1^* in the LP solution. This example is representative of how the LP solution can be quite sensitive to changes in agent 1's value. As is the case in these examples, the slope increases as agent 1's value increases.

8.6.2 Isolating Major Changes and Defining *get-sensitivity*

Having discussed the two ways in which a change in an agent's value affects a *node* in a search state, we present our implementation of *get-sensitivity*. The *get-sensitivity* method runs the search with agent 1's value set to w_1, but in addition to running the search, returns a higher value $w_1' > w_1$ at which to re-run the search. The guarantee is that if *get-sensitivity* returns w_1', then setting agent 1's value to any multiple of β in (w_1, w_1') and running BnB must still result in an allocation that contains agent 1.

As we run the BnB procedure, we can ask at each search state, what higher value of agent 1's report might cause a change to occur in the search? Therefore, we reduce *get-sensitivity* to the simpler problem of figuring out the value at which the decision at a single state would change. The minimum of these sensitivity values across all search states processed in BnB provides the next value to be returned by *get-sensitivity*. We call this single search state procedure *get-sens-single-state*.

A first attempt at *get-sens-single-state* would consider any higher value at which any aspect of a search state changes (e.g., the value of solution at any associated node in the state.) But this would trigger a large number of changes since the number of nodes scales with the number of steps in the search.

Instead, we focus (for a given state s) on identifying the next higher value at which the search decision changes (i.e., *whether or not we terminate, change the identity of the integral node in the case of termination, or change the selected node or branch variable if we do not terminate*).

In what follows, we assume that *select-node* is breadth-first and chooses the fractional node with highest value, although we can adapt the procedure to other choices of *select-node*. To handle other cases (such as depth-first), we have to add to the augmented search state the current deepest node whose value is not dominated by an integral node, and we need to add extra events to *breakpoint* which detect when this deepest node changes. In general, we need to modify the augmented search state and events detected to maintain the invariant that *breakpoint* returns a value w_1' such that values in $[w_{1\ temp}, w_1)$ all yield the same decision.

In order to find the lowest point where the decision associated with s changes, we introduce an augmented search state. Let $w_{1\ temp}$ be agent 1's value being currently considered. We make a distinction versus w_1 since $w_{1\ temp}$ can be a value higher than w_1, the value that we are currently examining for sensitivity purposes. The augmented search state adds the following information to a search state:

1. For each node in the state, compute the coefficient and bias for $w_{1\ temp}$.
2. Compute the best integral node for $w_{1\ temp}$.
3. Compute the best fractional node for $w_{1\ temp}$.

The purpose of this augmented search state is to allow us to understand how the search state changes as agent 1's value increases further from $w_{1\ temp}$, and thus when the search decision changes. As agent 1's value increases, we know that the values at the various nodes in the state will each increase linearly based on the coefficient of that node. With the assumption that *select-node* is breadth-first, the decision at s depends only on a comparison between the best integral node and the best fractional node. Figure 8.1(b) gives an example of the augmented search state.

The method *get-sens-single-state* for augmented state s' repeatedly finds the next lowest value $w_1' > w_{1\ temp}$ where one of the following changes occurs:

1. The best fractional node changes identity.
2. The best integral node changes identity.
3. The value of the best integral node crosses the value of the best fractional node (possibly multiplied by γ if we are not running to optimality).
4. The LP solution of some node changes.

For each such value, the method considers the next higher value on the discrete grid, and at this value checks to see whether the search decision would actually change at this state. If it does, then this becomes the relevant sensitivity value for this state— the first value at which the search decision first changes.

8.6.3 Correctness of *get-sens-single-state*

Theorem 8.6.1. *Method* get-sens-single-state *correctly computes the lowest point w_1' where the decision associated with search state s' would have changed.*

Proof. Let $w_{1\ temp}$ be as defined in *get-sens-single-state* and w_1' be the next lowest value where one of the events 1-4 occurs. We first show that s' is correct for all values in $[w_{1\ temp}, w_1')$. The coefficients and biases in s' are correct since event 4 has not triggered. The best integral and best fractional nodes are correct since 1 and 2 have not triggered. Since s' is correct for this range of values, the only way the decision could have changed

in this range of values is if the value of best integral node overtakes the value of the best fractional node or vice versa. This is caught by event 3, and so must not occur in the range $[w_{1\ temp}, w_1')$. Therefore, the decision is guaranteed not to change in $[w_{1\ temp}, w_1')$. Because we are discretizing, we can extend this range to $[w_{1\ temp}, w_{1\beta}')$ (the inputs to BnB have to be multiples of β). *get-sens-single-state* then recomputes the decision with the updated s' at $w_{1\beta}'$. If the decision has changed, then this is the lowest value at which a change occurs. If the decision does not change, then we can apply the same argument to show that the decision will not change in $[w_{1\beta}', w_1'')$, where w_1'' is the next value at which an event triggers.[13] □

***get-sens-single-state* Example**

Figure 8.1(b) shows the augmented search state s' for different ranges of w_1, agent 1's value. The LP solutions in nodes 2 and 5 are not dependent on w_1 while the solution in node 4 is dependent on w_1. When $w_1 \leq 0.5$, the LP solution in node 4 assigns $x_1^* = 0.5$ with a bias of 2.4. When $w_1 \geq 0.5$, the LP solution in node 4 assigns $x_1^* = 1.0$ with a bias of 2.15.

We now analyze *get-sens-single-state*. Suppose $w_1 = 0.25, \beta = 0.01$. Node 2 is the best integral node, while node 5 is the best fractional node. At $w_1' = 0.5$, event 4 is triggered as the LP solution in node 4 changes to a solution with $x_1^* = 1.0$. Note that at $w_1' = w_{1\beta}' = 0.5$, at node 4, the value of the previous LP solution is equal to the value of the new LP solution $(0.5 \cdot 0.5 + 2.4 = 0.5 + 2.15)$. No further updates are needed for s' as node 4's value (2.65) is still less than node 5's. Though the LP solution in node 4 has changed, the decision remains to select node 5 and continue searching. The same branch variable will be selected because the LP solution for node 5 has not changed. As a result, *get-sens-single-state* will continue, setting $w_{1\ temp}$ to 0.5. Assuming that no LP solutions change, the next event triggered will be event 1 at value $w_1' = w_{1\beta}' = 0.55$. At this value, node 4 will overtake node 5 as the best fractional node (the value of the LP solution in node 4 reaches 2.7 while the value of the LP solution in node 5 stays at 2.7 and we assume lexicographic tie-breaking by node index). The decision associated with s' will now change because node 4 will be selected as the next node to be explored. *get-sens-single-state* will return 0.55.

The example demonstrates the key ideas of *get-sens-single-state*. Not all events will lead to changes in the decision, but we need to capture all of these events to make sure that s'

[13]There is a technical detail here due to ties. If $w_1' = w_{1\beta}'$, then we have a point where the values of two nodes cross, and values will be tied at this point. As a result, $dec(s')$ at w_1' may differ from $dec(s')$ for $w_1' + \epsilon$ for a small ϵ. In these cases, we need to make sure to check the decision at the next multiple of β greater than w_1' even if events 1-4 do not occur before then.

reflects the true state if agent 1 were to report these higher values. In particular, event 4 is very important because it makes sure that the coefficient and bias values are valid for the range of agent 1's values being considered.

8.6.4 Implementing *get-sens-single-state*

The only problem that remains is how to compute the values when events 1-4 occur. For event 4, we saw in Section 8.6.1 how to detect the next highest value at which the LP solution changes. Because we detect event 4, we can assume that the coefficients and biases stored in s' are correct. For events 1 and 2, we need to make pairwise comparisons between the current best node and all other fractional and integral nodes respectively. For event 3, we need to compare the best fractional node against the best integral node. Events 1-3 then reduce to computing when the values of two nodes cross, given their coefficients and biases. If c_1, b_1, c_2, b_2 give the coefficients and biases of two nodes, we simply solve the following equation for the crossing value $w_1^* : c_1 w_1^* + b_1 = c_2 w_1^* + b_2 \Rightarrow w_1^* = (b_2 - b_1)/(c_1 - c_2)$.

8.6.5 Hot Restart and Inference

Hot Restart

With *get-sens-single-state*, we can now fully instantiate *get-sensitivity* and *optimized-is-deallocated-at-higher-values*. To check whether an agent becomes deallocated, take the minimum next value returned by calls to *get-sens-single-state* from every state in the BnB search and re-run BnB search with the agent's value updated to the minimum next value. This procedure may already outperform brute force sensitivity because we may skip over many higher multiples of β that would not have changed any search decision.

However, we can further improve performance with the following optimization. Suppose that the minimal next value w_1' returned by all the calls to *get-sens-single-state* across all decisions made in the search occurs at step 1000 in the search. This implies that the decisions at steps 1 through 999 would not have changed if agent 1's value is updated to w_1'. As a result, we need not re-run all these steps of the search. We can save the state after step 999 and rerun the search from this point. This inspires the following modified procedure for *get-sensitivity*.

Let $w_{1\ min}$ represent the lowest next-highest value returned by any call to *get-sens-single-state* thus far in the search. Whenever a search decision is made, *get-sens-single-state* is called. If the next value returned is weakly greater than $w_{1\ min}$, then ignore it (the search

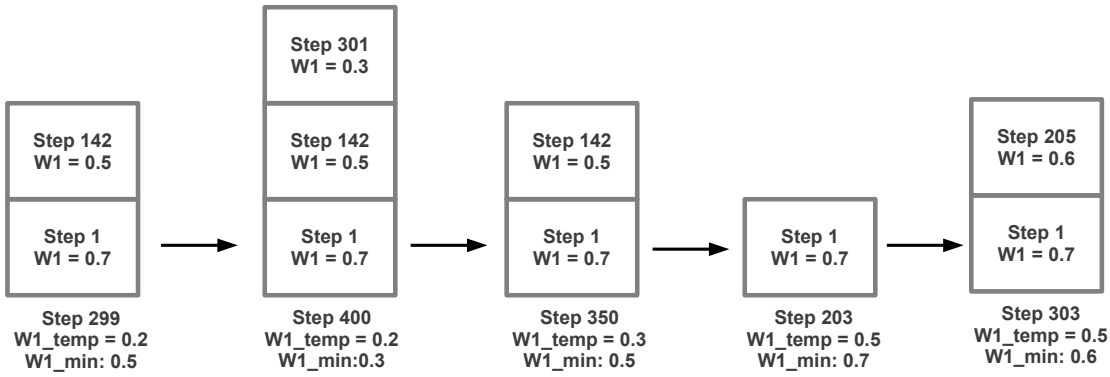


Figure 8.3: Progression of *optimized-is-deallocated-at-higher-values*.

decision would have changed earlier in the search). If the next value returned is less than $w_{1\ min}$, then reduce $w_{1\ min}$ to this value, and take a snapshot of the search state. Push this snapshot, along with $w_{1\ min}$, onto a list of search states from which to re-run. We refer to this as *hot restart*.

Figure 8.3 gives a way to view how this version of *optimized-is-deallocated-at-higher-values* proceeds. Each stack in the diagram represents the search states from which the search needs to be re-run based on current knowledge about the search, along with their starting steps and the associated sensitivity value for agent 1 in that state. Below each stack we give the step of the actual search, along with the current value ($w_{1\ temp}$) for agent 1 and the current minimum next value ($w_{1\ min}$) to which sensitivity checking will jump once the current search is complete. $w_{1\ min}$ will always equal the value stored in the top search state in the stack. As we proceed from left to right, we see that we might add search states to the stack. This occurs if *get-sens-single-state* returns a next value that is lower than $w_{1\ min}$. Once we have run a search state to completion, we process the next search state in the stack, running *get-sensitivity* starting at the indicated step and jumping agent 1's value $w_{1\ temp}$ forward to the stored value. This is possible because we take a snapshot of the search state whenever we add a search state to the stack. The stack will expand and shrink, but the current agent value $w_{1\ temp}$ will monotonically increase, and eventually, we will have processed all search states in the stack and completed sensitivity analysis for the agent.

Inference: Allowing Early-Stopping

Until this point, we are still actually running the search to completion (even though hot restart lets us start low in the tree) for all higher values that trigger a sensitivity check,

even though we only use the allocation to check whether the sensitivity agent remains in the allocation. This is all we care about: we don't need the full details of the allocation!

Leveraging this insight, we devise *early stopping rules* in the sensitivity checker. If we are sure that the search will terminate with a solution that contains agent 1, we do not need to run the search to completion (this is what we mean by "inference"). The main idea is to upper-bound the value of any solution where agent 1 is not allocated, and then use this upper bound to argue that the search will always terminate with a solution that allocates agent 1. One such upper bound is to take the max over the LP relaxations of all nodes in the current state, with the extra constraint $x_1 = 0$. This bound can be carried over from previous search states as well. Indeed, if the current best integral node allocates agent 1 and has value greater than these bounds, search can be terminated early. This is formalized in the following theorem.

Let s be the current search state. For a node k, let $k(x_1)$ indicate the value x_1^* in the solution stored at k. Let $LP_{x_i=0}(k)$ be the value of LP relaxation at a given node, with the extra constraint that variable $x_i = 0$.

Theorem 8.6.2. *Suppose that we are checking sensitivity for agent 1 at value w_1' and in a search state s. Suppose that the current best integral node in s allocates agent 1 and has value w.*

1. *Let*

$$w' = \max_{k:k\in F(s)} LP_{x_1=0}(k).$$

If $w' < w$, then the search starting from s will result in a solution that allocates agent 1.

2. *Let s' be any search state that has been run to completion (i.e. all fractional nodes have been pruned by some leaf node). Let*

$$w' = \max_{k:k\in F(s')} LP_{x_1=0}(k),$$
$$w'' = \max_{k:k\in I(s'),k(x_1)=0} val(k).$$

If $\max(w', w'') < w$, then the search starting from s will result in a solution that allocates agent 1.

Proof. The main idea of the proof is to show that w' in the first case and $\max(w', w'')$ in the second case are valid upper bounds on the value of any solution with $x_1 = 0$. Once this

is shown, we know that the search starting in s will not find any solutions that set agent 1 to 0 and have better solution value than the current best solution, which sets agent 1 to 1.

In the first case, all leaf nodes with $x_1 = 0$ have value less than w and will not be selected. The children of the fractional nodes may yield a solution with $x_1 = 0$, but the value of these solutions is upper bounded by the value of the relaxation, $LP_{x_1=0}(k)$.

In the second case, the key observation is that the value of a solution with $x_1 = 0$ does not change as agent 1's value increases. As a result, we can use upper bounds on solutions with $x_1 = 0$ from other search states s' that have already been run to completion. w' gives the upper bound across fractional nodes in s', while w'' gives the upper bound across leaf nodes with $x_1 = 0$. Any solution with $x_1 = 0$ is either a leaf node with $x_1 = 0$ or a child of an fractional node, and therefore $\max(w', w'')$ is a valid upper bound on the value of any solution with $x_1 = 0$. □

At the cost of some extra computation (computing LP relaxations with the $x_1 = 0$), this allows us to stop searching once it is clear agent 1 will be allocated in any solution returned.

8.6.6 Linear Program Caching, Parallelization

Linear Program Caching

The most expensive part of BnB search and sensitivity analysis is solving the LP relaxations for nodes. However, a key insight is that in the course of sensitivity analysis, we may revisit nodes with the same integer partial assignment over and over, with the only difference being that w_1 might be set to a higher value. As a result, when running *get-sensitivity*, we cache LP solutions, along with the upper bounds for when the LP solutions change (as in Section 8.6.1). When we need to solve an LP in a later BnB search with value w_1', we first make a lookup in this cache to see if there is an already computed LP solution whose upper bound is greater than w_1' and reuse the previously computed solution if one is found. LP solves dominate the runtime for sensitivity analysis, and this greatly decreases the number of solves needed for sensitivity analysis (see Section 8.8.2).

An optimization related to LP caching is that of using optimal solutions from parent nodes in the BnB search tree to "hot start" the LP solve process for child nodes during sensitivity analysis. We did implement this, but we did not see substantial gains so we abandoned it for simplicity and to keep our memory footprint small. It would be of interest to pursue this direction further in future work.

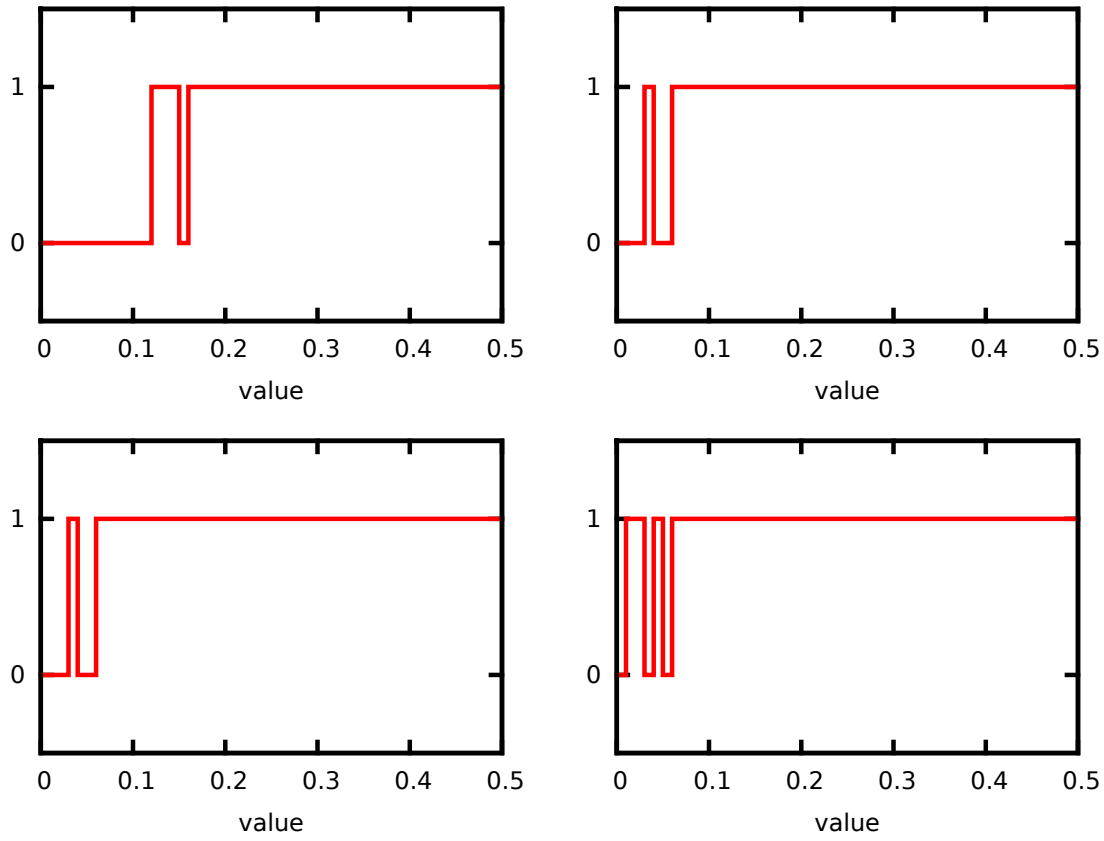


Figure 8.4: The allocation function for different agents on a decay instance with 1000 agents and 100 items (see Section 9.7). Agents were chosen because of non-monotonicities in the BnB search for these agents.

Parallelization

While we have to perform sensitivity analysis for every allocated agent, the sensitivity analysis for each agent is completely independent of the sensitivity analysis for other agents. As a result, sensitivity checking can be perfectly parallelized. In our experimental results, we report this parallelized runtime, which is the time required to solve the initial search plus the maximum runtime for *optimized-is-deallocated-at-higher-values* across all allocated agents.

8.7 Making Branch-and-Bound Search more Monotone

In order for the allocation computed by *discretized-ironed-alloc* to have good welfare properties, we need the underlying heuristic algorithm to be monotone for many agents on many instances. If not, then many agents will be deallocated, and even if the original, un-ironed

solution has high welfare, the ironed solution will not. Recognizing this, we introduce two methods for making BnB more monotone.

8.7.1 Input Discretization

As discussed in Section 8.4, one way to decrease the number of deallocations is to increase the grid size β. With discretization, an agent remains allocated as long as the heuristic allocation function continues to allocate the agent for all higher multiples of β. Figure 8.7 shows the allocation curve for several agents in one of our experimental instances. The figure is generated using $\beta = 0.01$. Many of the non-monotonicities in the curves survive for a small range of values. Increasing β allows these small ranges to be skipped over and increases monotonicity. But there is a tradeoff with solution quality because the input is approximated.

8.7.2 Fractional Bucketing

We also introduce the notion of "bucketing" when selecting which variable to branch at a node. The classic variable selection algorithm in BnB search is to take the most fractional variable; i.e., the variable with value closest to 0.5. But this is very sensitive to small changes in the LP solution, and can result in many search decision changes even if the selected node remains the same since the branch variable may change. To remedy this, we experiment with bucketing variables based on their fractionality and choosing the lexicographically first variable in the smallest bucket. For example, consider an LP solution $\{x_1 = 0.41, x_2 = 0.48, x_3 = 0.7, x_4 = 0.51\}$. The most fractional variable without any bucketing is x_4. But with a bucket size of 0.2, x_1, x_2, x_4 are all placed in the same bucket (the bucket representing values in $[0.4, 0.6]$), and we break ties on x_1. In the extreme case of a bucket size of 1.0, all variables belong to the same bucket, but we make the exception that we don't select variables that are already set to 0.0 or 1.0; therefore, a bucket size of 1.0 amounts to selecting the first variable that is set to a non-integer value. Larger bucket sizes make the underlying search more monotone since the decisions in the search are less sensitive to small changes in the LP solutions, and we see this in our experimental results.

8.8 Experimental Results

In this section, we present experimental results based on an implementation of monotone BnB search for known single-minded CAs. Our experiments are performed using a custom

Java implementation of BnB search, using CPLEX as our LP solver. The experiments are run on a machine with two 8 core 2.4GHz Intel Xeon processors. We implement the optimized version of *get-sensitivity*, as well as hot restart, inference / early-stopping, and LP caching.

We generate agent valuations using the *decay* (L4) distribution with parameter $\alpha = 0.75$ and a number of agents equal to ten times the number of items as this has been shown to generate hard winner determination instances [Leyton-Brown et al., 2000, Sandholm et al., 2005]. In our experiments, we fix node selection to choose the deepest node if no integral node has been found, and the node with highest value otherwise. For variable selection, we select the most fractional variable with different bucket sizes, as described in Section 8.7.2. For our discretization procedure, we first normalize values to $[0, 1]$ by dividing by a maximum value. For the L4 distribution with $\alpha = 0.75$, 30 is a reasonable maximum value. Values larger than 30 are normalized to 1.0.

8.8.1 Welfare Analysis

We generate 50 random instances from the decay distribution with 300 agents, 30 items, and $\alpha = 0.75$. We vary γ, β, and the variable selection algorithm. For variable selection, the bucket sizes that we consider are *no bucket size*, 0.2 and 1.0. For this dataset, running to full optimality is very fast (on the order of seconds), so these instances do not represent a domain on which we would want to use our ironing procedure. Rather, they are a way to examine the impact of search parameterization on the quality of the ironed solution.

Figure 8.5 presents the welfare results. Each graph is for a particular $\gamma \in \{0.9, 0.95, 0.99\}$ and plots average welfare of the solution across the 50 instances (relative to the optimal) as β increases (i.e., more discretization.) *greedy-LOS* indicates the welfare of the greedy algorithm from Lehmann et al. [2002], while *greedy-5* indicates the welfare of the algorithm from Mu'alem and Nisan [2008] with a parameter choice of 5. Beyond this value, the runtime becomes prohibitive without much improvement in welfare (Figure 8.6). The *orig* line indicates the *original welfare* of the solution, i.e. welfare before we check whether agents need to be deallocated. The *iron* line indicates the *ironed welfare*, i.e. welfare after agents have been deallocated. To make the plots clearer, we plot the original welfare for the most fractional variable without bucketing. The original welfare is similar for the other bucketing strategies. Lines with b followed by a floating point number indicate use of bucket sizes. For instance, *b0.2-iron* plots the ironed welfare for bucket size 0.2.

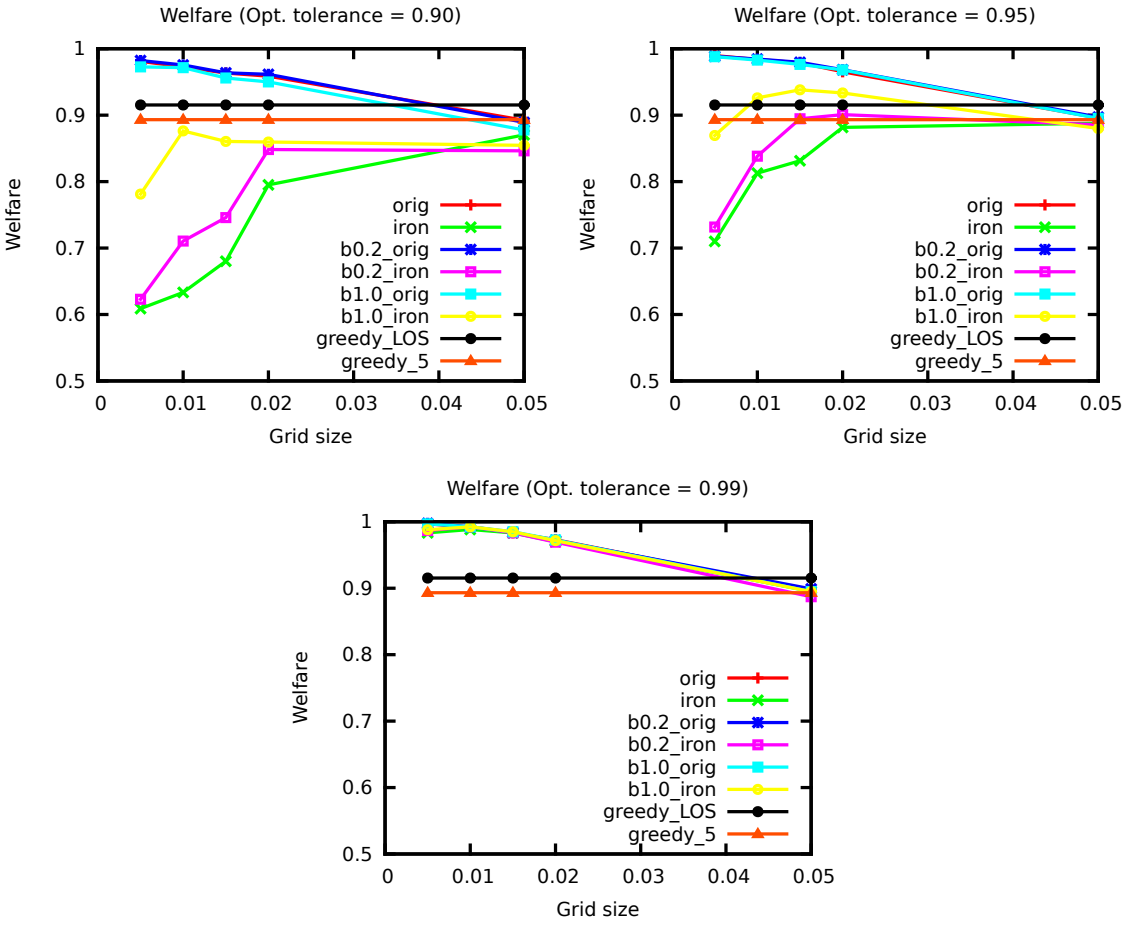


Figure 8.5: Average welfare (compared to the optimal) for different search parameterizations on small instances.

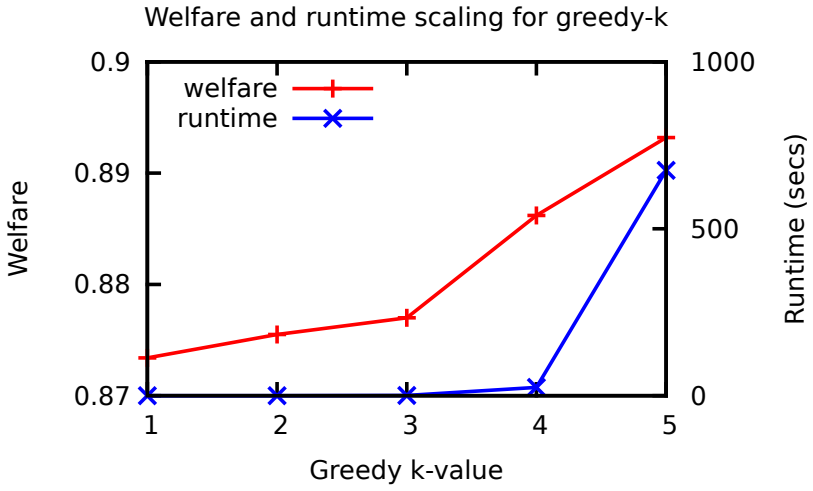


Figure 8.6: Welfare and runtime for different choices of k for $MAX(Exhaustive(k), G_k)$ algorithm in Mu'alem and Nisan [2008]. $Exhaustive(k)$ is exhaustive search over allocations of size k, and G_k is a greedy algorithm that ranks using the compact ranking with parameter k. In the compact ranking with parameter k, a bundle gets a score equal to its value if its size is at most $\sqrt{m/k}$ and 0 otherwise. This algorithm gives an $O(\epsilon\sqrt{m})$ worst-case approximation to the optimal social welfare, where the choice of k depends on ϵ.

Grid size (β)

Figure 8.5 illustrates the effect of the grid size, β, on the welfare of the ironing algorithm. If β is too small, then there are many deallocations, and ironed welfare suffers. If β is too large, then optimizing against the discretized values gives a poor approximation to the original problem, and welfare suffers. There is a peak grid size that is optimal.

Most fractional bucket size

Figure 8.5 confirms that fractional-variable bucketing has a positive effect on the monotone BnB. The curves with the highest ironed welfare are those with bucket size 1.0.

Optimality tolerance (γ)

The original welfare is quite similar across different values of γ. We also see that the welfare of the ironed solution improves as γ increases. In particular, for $\gamma > 0.95$, an optimally parameterized ironing algorithm does better than the greedy algorithms. (*greedy-LOS* and *greedy-5* do not have a grid size, so they appear as a constant line.) This occurs with $\gamma = 0.95$ despite monotonicity failure and agent deallocations. The good performance is not because the underlying "orig" algorithm is identifying optimal solutions (and thus

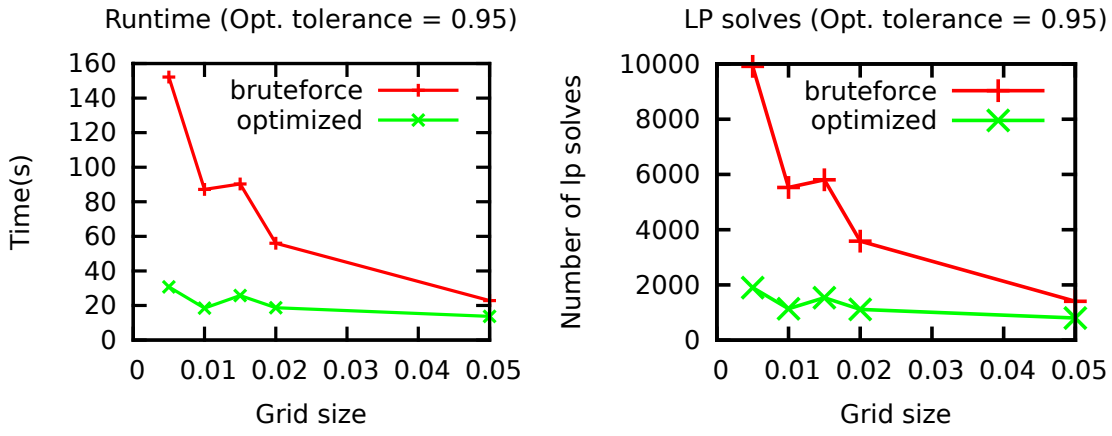


Figure 8.7: Runtime and number of LP solves for brute force and optimized ironing

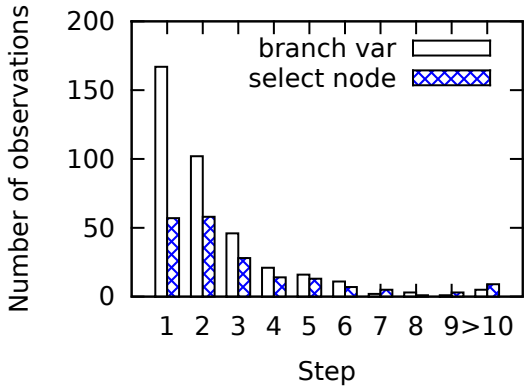


Figure 8.8: Histogram of the steps at which search decisions change, for different change types. $\gamma = 0.95, \beta = 0.01$, averaged over all bucket sizes and datasets. 300 agents, 30 items.

monotone). Averaged across all bucket sizes, for $\beta = 0.01$ and $\gamma = 0.9$, 34% of the original solutions are optimal (with respect to the particular grid size). This increases to 45% and 92% for $\gamma = 0.95$ and $\gamma = 0.99$.

8.8.2 Effectiveness of Optimized Sensitivity

Comparison to brute force

We compare the brute force approach with the optimized sensitivity approach. We label the sensitivity checking procedure for BnB the "optimized" algorithm. Figure 8.7 plots the runtime and number of LPs solved across different grid sizes for $\gamma = 0.95$ and no bucket size. The graphs for different bucket sizes look very similar so for clarity, we only display no bucket size.

It shows that runtime is highly correlated with the number of LP solves. The optimized ironing procedure yields the biggest gains when the grid size is small, though for all grid sizes, the optimized procedure does have better runtime and a smaller number of calls to the LP solver. Brute force checks every higher multiple of the β and thus performs work linear in 1 / β. Even for large grid sizes, where the brute force procedure only needs to make a small number of calls to check sensitivity, the optimized procedure appears to match or slightly improve on its performance. This is likely due to the fact that the optimized procedure also leverages larger β in that it rounds to the next highest multiple of β when checking sensitivity. We did not implement LP caching for the brute force trials, though, which could potentially decrease runtime and number of LP solves.

LP caching

We define an LP lookup as any time during sensitivity checking when we request the LP solution at a given value for the sensitivity agent. LP caching allows us to reuse previously computed solutions. With $\gamma = 0.95, \beta = 0.01$, and grouping all bucket sizes together, the average cache hit rate for LP lookups is around 30%.

8.8.3 Analysis of Search Changes

We also study the particular types of decision changes that take place during sensitivity checking. For $\gamma = 0.95, \beta = 0.01$, most decision changes are branch variable changes rather than select node changes. With no bucket size, there is an average of 27.1 branch variable changes and 3.8 select node changes. For bucket size 1.0, these numbers decrease to 15.4 and 8.0, indicating that larger bucket sizes do decrease the number of branch variable changes. Figure 8.8 examines when these changes occur, and we see that branch variable changes tend to occur in the earlier steps of the search while select node changes are more evenly distributed.

8.8.4 Hard Instances

In the previous section, the instances we test are easily run to optimality by BnB in a few seconds. However, they provide useful insights into the parameterizations of our framework and show that our algorithm can outperform the welfare of greedy algorithms. In this section, we examine instances where optimal BnB is more computationally intensive and takes minutes to run to completion. We use decay instances with 1000 agents and 100 items, $\alpha = 0.75$. We test different parameters, but in Tables 8.1 and 8.2, we focus on the best

Table 8.1: Welfare (% of optimal) on hard instances.

greedy (LOS)	orig-0.025-98-1.0	iron-0.025-98-1.0
0.94	0.96	0.96 (+0.02)
0.89	0.93	0.93 (+0.04)
0.92	0.93	0.93 (+0.00)
0.94	0.93	0.93 (-0.01)
0.87	0.92	0.92 (+0.05)
0.91	0.93	0.93 (+0.02)
0.89	0.93	0.92 (+0.03)
0.91	0.91	0.91 (+0.00)
0.88	0.93	0.93 (+0.04)
0.92	0.93	0.93 (+0.01)

Table 8.2: Runtime (minutes) on hard instances

optimal	0.025-98-1.0 (t)	0.025-98-1.0 (p)
4.55	0.41	0.03
0.24	0.02	0.00
1.17	0.14	0.01
0.53	0.10	0.01
1.94	0.14	0.01
0.78	0.02	0.00
2.25	0.43	0.03
0.47	0.39	0.02
1.26	2.11	0.14
0.20	0.17	0.01

performing parameters $\beta = 0.025, \gamma = 0.98$ with bucket size 1.0. For welfare, *orig* indicates the pre-ironed welfare, while *iron* indicates the welfare after deallocations. For runtime, t indicates the total runtime for monotone BnB, while p indicates the fully parallelized runtime discussed in Section 8.6.6.

From Table 8.1, we see that the welfare produced is better than greedy on these hard instances, and also that few agents are deallocated as the ironed welfare is close to the original welfare. We only report *greedy-LOS* since it outperforms parameterizations of *greedy-k* for values of k with runtimes comparable to monotone BnB. Table 8.2 gives the runtime for optimal BnB and the total and parallelized runtime for monotone BnB. Running to optimality tends to take more time than monotone BnB, but there are exceptions. In addition, the fully parallelized runtime (Section 8.6.6) for our algorithm is better than optimal BnB. We also note that to maintain truthfulness with optimal BnB we must be able to run every instance to completion, so we care about the long tail of the runtime distribution. With monotone BnB, the search itself is fast because we run to an optimality

tolerance, and sensitivity checking for a single agent is not overly expensive. The expense comes in having to check every allocated agent, and as we have mentioned, this can be parallelized. In this sense, we can reliably decrease runtime for monotone BnB with more computational resources, in contrast with the scalability that would be offered by state-of-the-art parallel Branch-and-Bound solvers.

8.9 Summary and Future Work

We introduce a method for monotone BnB search by performing automated sensitivity analysis in regard to changes in the outcome of search in response to changes in objective value coefficients. In application to known single-minded CAs, the experimental results demonstrate higher decision quality compared with greedy algorithms when coupled with an optimality tolerance $\gamma < 1$, and thus when the algorithm is not simply identifying the optimal allocation. We believe the results in regard to the scalability of sensitivity checking of BnB search are promising, and given the generality of the approach, hope to uncover additional optimizations. Possible areas for further improvement are additional inferential approaches that allow for short-circuiting, as well as additional ways to encourage monotonicity. We may also be able to leverage the fact that LP value, as a function of a specific agent's value, is convex.

8.9.1 Future work

1. Basic to our approach is the idea of performing sensitivity analysis for a given input and adjusting the algorithm *on that input* so that the algorithm is monotone over the entire input space. Probably the most intriguing, and challenging, direction for future work is to understand whether this local adjustment is possible in achieving appropriate notions of monotonicity in problems of multi-dimensional mechanism design.

2. Extend monotone BnB search to other single-dimensional mechanism design problems, including non-downward closed environments (e.g., scheduling, where correcting a failure of monotonicity could involve introducing additional "dummy" jobs for a machine to process).

3. Explore the idea of sensitivity and computational ironing on other methods of heuristic search, for example local search.

4. Extend monotone BnB search to handle cut generation, and expose a parameterized

search framework to the methods of empirical algorithm design, to allow for automated configuration [Hutter et al., 2010].

5. Consider alternative methods to "correct" an allocation when a failure of monotonicity is identified, for example introducing randomization to allow for smoother notions of monotonicity.

6. Related to this, for domains such as scheduling where downward closed does not hold, consider output ironing by checking the sensitivity of the allocation of jobs to a machine for higher costs, and if we find more work at a counter-factual, allocating the present agent more work by using dummy jobs (that is, fix by adding work rather than removing resources.).

Chapter 9

Learning Payment Rules

We have thus far limited our discussion of mechanism design to incentive compatible mechanisms. That is, the goal is to identify a mechanism that satisfies the incentive constraints and optimizes a given design objective such as welfare or revenue. Chapter 6 concerns such a mechanism design problem in the context of cake cutting and Chapter 8 examines this problem for known single-minded CAs.

There are, however, significant challenges associated with this classical approach. First, it can be analytically cumbersome to derive well-performing DSIC or BIC mechanisms for domains that are multi-dimensional, in the sense that each agent's private information is described through more than a single number. An example of a multi-dimensional domain is a multi-minded CA. Multi-minded CAs generalized single-minded CAs in that agents can have up to k target bundles. Single-minded CAs are also technically multi-dimensional as agent preferences are described by their target bundle and their value for that bundle, but multi-minded CAs offer a clearer case and will be studied in this chapter. Though we can use the VCG mechanism to optimize welfare for multi-minded CAs, there are few results for working with other objective functions. Additionally, if we impose computational constraints then we have few positive results. The positive results for single-minded CAs discussed in Section 7.4 do not naturally extend to multi-minded CAs.

Second, incentive-compatibility can be costly, in that adopting it as a hard constraint can preclude mechanisms with other desirable properties. For example, imposing DSIC necessarily leads to poor revenue, vulnerability to collusion, and vulnerability to false-name bidding in CAs where valuations exhibit complementarities among items [Ausubel and Milgrom, 2006, Rastegari et al., 2011]. Of course, by the revelation principle (Section 2.2.2) this weakness should be ascribed to insisting on mechanisms that are analyzed in equilibrium

(dominant-strategy or otherwise) and not to the imposition of incentive constraints *per se*.

9.1 Our Results

In the face of these difficulties, we adopt statistical machine learning to automatically infer mechanisms with good incentive properties. Rather than imposing incentive-compatibility as a hard constraint, we start from a given outcome rule, typically expressed as an algorithm, and then use machine learning techniques to identify a payment rule that minimizes agents' *expected ex post regret*. The ex post regret (or just *regret* where it causes no confusion) of an agent for truthful reporting in a given instance is the maximum amount by which its utility could increase through a misreport holding constant the reports of others. The expected ex post regret is the average ex post regret over all agents and all preference types, calculated with respect to a distribution on types.

While a mechanism with zero regret for all agents on all inputs is strategyproof, we are especially interested in settings where the outcome rule does not allow for exact incentive-compatibility. In this sense, the approach adopted in this paper is not an equilibrium approach. But, there are two important comments to make in this regard.

First, we insist that an agent's payment, conditioned on an outcome, is independent of its report. The only way an agent can improve its utility is by changing its report in a way that changes the outcome. Generically, this ensures mechanisms that provide zero marginal benefit to deviation from truthful reports. This property is seen in practice in the *generalized second-price* auction (GSP) used for sponsored search. This local stability property has been emphasized by Erdil and Klemperer [2010] in the context of CA design.

In addition, a bound on expected regret implies a bound of the form "interim regret is at most ϵ with probability at least 1-δ," where interim regret is the ex post regret to an agent for a particular type, averaged over all types of other agents. Based on this, support for expected regret can be developed through a simple model of costly manipulation, where agents face some cost for trying to engage in strategic behavior, and choose not to engage in manipulation if this cost is greater than the ϵ-bound on interim regret. In this case, a $1-\delta$ fraction of the interim agents will have no incentive to manipulate, though there remains a δ fraction for whom the benefit to manipulation may be greater than ϵ.

Our approach is applicable to domains that are multi-dimensional, and for domains for which the computational efficiency of outcome rules is a concern. In particular, we are interested in domains for which incentive-compatibility is unavailable or undesirable,

given the implications imposed on outcome rules (e.g., requiring that outcome rules have undesirable economic or computational properties.) Because the payment rule is learned on the basis of a given outcome rule, the framework is most meaningful in domains where revenue considerations are secondary to properties of outcome rules.

The essential insight that underpins our approach is that the payment rule of a strategyproof mechanism can be thought of as a classifier for predicting the outcome. In particular, the payment rule implies a price to each agent for each outcome, and the selected outcome in a mechanism must simultaneously maximize the reported value minus price for every agent. The discriminant function of a classifier provides a score to different outcomes for a given input, with the outcome with the highest score corresponding to the prediction of the classifier. By limiting classifiers to discriminant functions with this "value-minus-price" structure, where the price can be an arbitrary function of the outcome and the reports of other agents, we obtain a remarkably direct connection between multi-class classification and mechanism design.

For an appropriate loss function, the discriminant function of a classifier that minimizes generalization error over a hypothesis class has a corresponding payment rule that minimizes expected ex post regret among all payment rules corresponding to classifiers in this class. Conveniently, an appropriate method exists for multi-class classification with large outcome spaces that supports the specific structure of the discriminant function, namely the method of *structural support vector machines* [Tsochantaridis et al., 2005, Joachims et al., 2009]. Just like standard support vector machines, this also allows us to adopt non-linear kernels, thus enabling discriminant functions and thus price functions that depend in a non-linear way on the outcome and the reported types of agents.

The computational cost associated with our approach occurs offline during training, which is the process of learning a payment rule for a given outcome rule. The learned payment rules are fast to evaluate at run-time, i.e. in the context of a deployed mechanism, and have a succinct representation through the standard support-vector machine approach. A challenge in structural support vector machines is to handle the large number of possible outcomes (i.e., labels in the classification problem) during training.

One way to address this in our context is to work with valuation functions for which the training problem can be formulated as a succinct, convex optimization problem. In particular, we adopt k-wise dependent valuations Conitzer et al. [2005] and leverage a connection with maximum *a posteriori* (MAP) assignment for Markov networks to scale-up our framework in application to CAs.

In illustrating the framework, we focus on three situations where strategyproof payment rules are not available:

(i) multi-minded combinatorial auctions, in which each agent is interested in a constant number of bundles, where winner determination is provided through a greedy allocation rule,

(ii) an assignment problem with multiple distinct items and agents with unit-demand valuations and an *egalitarian* outcome rule, i.e., an outcome rule that maximizes the minimum value of any agent, and

(iii) combinatorial auctions with k-wise dependent valuations, in which each agent's valuation has a graphical representation and winner determination is provided through a greedy allocation rule.

The experimental results demonstrate low expected regret even when the 0/1 classification accuracy is only moderately good, and better regret properties than those obtained through the simple VCG-based payment rules that we adopt as a baseline. In addition, we give special consideration to the failure of ex post individual rationality, and introduce methods to bias the classifier to avoid these kinds of errors and also *post hoc* methods to adjust trained payments, or even allocations, to reduce or eliminate them.

For setting (i), we find that our learned rules perform similarly to VCG-based rules. In setting (ii), our learned rules perform significantly better than VCG-based rules, which is understandable given that the egalitarian objective is quite different from the welfare maximizing objectives to which the VCG idea is designed. In setting (iii), our learned rules provide better regret properties than VCG-based rules for large numbers of items, and allow us to trade-off between IR violation and regret more effectively than VCG-based rules. While our experiments for CAs in setting (i) are limited to only five items, we are able to scale to instances with tens of items in setting (iii) as our training problem is polynomial in the number of items even though we are running a combinatorial auction.

9.2 Related Work

Our work is related to computational approaches to mechanism design discussed in Section 1.2.3. The main difference between our approach and the automated mechanism design (AMD) approach [Conitzer and Sandholm, 2002, Guo and Conitzer, 2010b, Cai et al., 2012a] is that we assume that we are provided an outcome rule and seek just the payment rule, whereas AMD approaches optimize over both rules simultaneously. These approaches

seem limited to domains in which the outcome rule can be succinctly represented, which likely is not the case for the kinds of combinatorial auction problems that we consider in this chapter.

Our work is also related to the approaches discussed in Section 1.2.3 that convert approximation algorithms into truthful mechanisms for different notions of truthfulness. A key difference between our work and this literature is that we do not enforce conditions on the provided outcome rule whereas this literature is focused on welfare maximization settings. Additionally, the target of minimizing expected ex post regret and the imposition of agent-independent prices make the incentive properties of mechanisms designed through our approach incomparable to mechanisms that are truthful in expectation or BIC. In comparison to mechanisms that are truthful in expectation which are necessarily randomized, our mechanisms are deterministic.

In comparison to BIC, ex post regret is stronger in that BIC assumes that an agent can misreport to a single other type for all type reports of other agents while ex post regret looks at each possible realization of types for other agents and allows the agent to make a different report for each realization. On the other hand, ϵ expected ex post regret may not guarantee ϵ-BIC since we take the expectation over an agent's own type, so the interim regret could be larger than ϵ for certain realizations of an agent's type. However, as mentioned earlier, given the expected ex post regret we can derive a bound of the form "interim regret is at most ϵ' with probability at least 1-δ."

Finally, in determining the outcome and payments for a given instance, the approach of Bei and Huang [2011] and Hartline et al. [2011] evaluates the outcome rule on a number of randomly perturbed replicas of that instance that is polynomial in the number of agents, the desired approximation ratio, and a notion capturing the complexity of the type spaces. When type spaces are large, as in the case of CAs, this may become intractable. By contrast, our approach evaluates the outcome rule and the trained payment rule once for a given instance and incurs additional computational costs only during training.

The work of Lahaie [2009, 2010] precedes our work in adopting a kernel-based approach for combinatorial auctions, but focuses not on learning a payment rule for a given outcome rule but rather on solving the winner determination and pricing problem for a given instance of a combinatorial auction. Lahaie introduces the use of kernel methods to compactly represent non-linear price functions, which is also present in our work, but obtains incentive properties more indirectly through a connection between regularization and price sensitivity. The main distinction between the two lines of work is that Lahaie focuses on the design of

scalable methods for clearing and pricing approximately welfare-maximizing combinatorial auctions, while we advance a framework for the automated design of payment rules that provide good incentive properties for a given outcome rule, which need not be welfare-maximizing.

Our discussion of k-wise dependent valuations builds on valuation structure for combinatorial auctions introduced by Conitzer et al. [2005] and Abraham et al. [2012]. Our tractable training results rely on connections between k-wise dependent valuations and associative Markov networks [Taskar et al., 2004].

A discussion of related work on approximate incentive compatibility, or incentive compatibility in the large market limit, can for example be found in the recent surveys by Carroll [2011] and Lubin and Parkes [2012]. A fair amount of attention has been devoted to regret-based metrics for quantifying the incentive properties of mechanisms [e.g., Parkes et al., 2001, Day and Milgrom, 2008, Lubin, 2010, Carroll, 2011]. Pathak and Sönmez [2010] provide a qualitative ranking of different mechanisms without payments in terms of the number of manipulable instances. Budish [2010] introduces an asymptotic, absolute design criterion regarding incentive properties in a large replica economy limit. Lubin and Parkes [2009] provide experimental support that relates the divergence between the payoffs in a mechanism and the payoffs in a strategyproof "reference" mechanism and the amount by which agents deviate from truthful bidding in the Bayes-Nash equilibrium of a mechanism.

9.3 Preliminaries

We adopt the same notation as that of Section 2.2. Specifically, we assume a mechanism design with money setting, and assume that a mechanism is defined by a pair of functions (g, p), where $g : \Theta \to \Omega$ and $p : \Theta \to \mathbb{R}^n_{\geq 0}$. Recall that when appropriate, $g_i(\theta)$ indicates the part of the outcome that agent i receives (e.g., the items allocated to agent i in a combinatorial auction). We will consider settings without externalities and outcome rules that satisfy consumer sovereignty and reachability of the null outcome (Section 2.2.1). Theorem 2.2.10, which gives a simple characterization of strategyproofness, is crucial for our approach as it provides the basic insight into how to utilize the discriminant function of a classifier as a payment rule.

We quantify the degree of strategyproofness of a mechanism in terms of *ex post regret* (see Definition 2.2.2 in Section 2.2) which quantifies the maximum amount an agent can

gain by misreporting its type.

Analogously, the *ex post violation of individual rationality* of agent $i \in N$ in mechanism (g, p), given true type $\theta_i \in \Theta_i$ and reported types $\theta'_{-i} \in \Theta_{-i}$ of the other agents, is

$$irv_i(\theta_i, \theta'_{-i}) = |\min(u_i((\theta_i, \theta'_{-i}), \theta_i), 0)|.$$

This quantity is zero when there is no violation of individual rationality (IR) for the agent at this type profile, but negative when the agent's utility is negative for the outcome and payment.

We consider situations where type profiles θ are drawn from a distribution with probability density function, $D : \Theta \to \mathbb{R}$, such that $D(\theta) \geq 0$ and $\int_{\theta \in \Theta} D(\theta) = 1$. Given such a distribution, and assuming that all agents report their true types, the *expected ex post regret* of agent $i \in N$ in mechanism (g, p) is $\mathbb{E}_{\theta \sim D}[rgt_i(\theta_i, \theta_{-i})]$.

Outcome rule g is *agent symmetric* if for every permutation π of agents N, and all types $\theta, \theta' \in \Theta$ such that $\theta_i = \theta'_{\pi(i)}$ for all $i \in N$, $g_i(\theta) = g_{\pi(i)}(\theta')$ for all $i \in N$. This specifically requires that $\Theta_i = \Theta_j$ and $\Omega_i = \Omega_j$ for all $i, j \in N$. Similarly, type distribution D is *agent symmetric* if $D(\theta) = D(\theta')$, for every permutation π of N, and all types $\theta, \theta' \in \Theta$ such that $\theta_i = \theta'_{\pi(i)}$ for all $i \in N$. Given agent symmetry, a price function $t_1 : \Theta_{-1} \times \Omega_i \to \mathbb{R}$ for agent 1 can be used to generate the payment rule p for a mechanism (g, p), with

$$p(\theta) = \big(t_1(\theta_{-1}, g_1(\theta)), t_1(\theta_{-2}, g_2(\theta)), \ldots, t_1(\theta_{-n}, g_n(\theta))\big),$$

so that the expected ex post regret is the same for every agent.

We assume agent symmetry going forward, which precludes outcome rules that break ties based on agent identity, but obviates the need to train a separate classifier for each agent while also providing some benefits in terms of simplifying the presentation of our results. The experimental results are not affected by this assumption because ties occur only with negligible probability. However, our framework can handle settings where either the outcome rule is not agent symmetric or the type distribution is not agent symmetric. In these cases, we would need to solve a separate training problem for each agent and learn an agent-specific payment rule.

9.4 Payment Rules from Multi-Class Classifiers

A *multi-class classifier* is a function $h : X \to Y$, where X is an input domain and Y is a discrete output domain. One could imagine, for example, a multi-class classifier that labels a given image as a dog, cat, or some other animal. In the context of mechanism design, we will be interested in classifiers that take as input a type profile and output an outcome. What distinguishes this from an outcome rule is that we will impose restrictions on the form the classifier can take.

Classification typically assumes an underlying target function $h^* : X \to Y$, and the goal is to learn a classifier h that minimizes disagreements with h^* on an input distribution D_X on X, based only on a finite set of *training data* $\{(x^1, y^1), \ldots, (x^\ell, y^\ell)\} = \{(x^1, h^*(x^1)), \ldots, (x^\ell, h^*(x^\ell))\}$ with $x^1, \ldots, x^\ell$ drawn from D_X. This may be challenging because the amount of training data is limited, or because h is restricted to some hypothesis class $\mathcal{H}$ with a certain simple structure, e.g., linear threshold functions. If $h(x) = h^*(x)$ for all $x \in X$, we say that h is a *perfect classifier* for h^*.

We consider classifiers that are defined in terms of a *discriminant function* $f : X \times Y \to \mathbb{R}$, such that

$$h(x) \in \arg\max_{y \in Y} f(x, y)$$

for all $x \in X$. More specifically, we will be concerned with *linear* discriminant functions of the form

$$f_w(x, y) = w^T \psi(x, y)$$

for a weight vector $w \in \mathbb{R}^b$ and a *feature map* $\psi : X \times Y \to \mathbb{R}^b$, where $b \in \mathbb{N} \cup \{\infty\}$. The function ψ maps input and output into an b-dimensional space, which allows non-linear features to be expressed. In general, we allow w to have infinite dimension, while requiring the inner product between w and $\psi(x, y)$ to remain well-defined. Computationally, the infinite-dimensional case is handled through kernels, as described in Section 9.5.1.

9.4.1 Mechanism Design as Classification

Given an outcome rule g and access to a distribution D over type profiles, our goal is to design a payment rule p that gives the mechanism (g, p) the best possible incentive properties, in the sense of expected regret.

Assuming agent symmetry, we focus on a partial outcome rule $g_1 : \Theta \to \Omega_1$ and train a classifier to predict the outcome to agent 1. To train a classifier, we generate examples by

drawing a type profile $\theta \in \Theta$ from distribution D and applying outcome rule g to obtain the target class $g_1(\theta) \in \Omega_1$.

We impose a special structure on the hypothesis class. A classifier $h_w : \Theta \to \Omega_1$ is *admissible* if it is defined in terms of a discriminant function f_w of the form

$$f_w(\theta, o_1) = w_1 v_1(\theta_1, o_1) + w_{-1}^T \psi(\theta_{-1}, o_1)$$

for weights w such that $w_1 \in \mathbb{R}_{>0}$ and $w_{-1} \in \mathbb{R}^m$, and a feature map $\psi : \Theta_{-1} \times \Omega_1 \to \mathbb{R}^m$ for $b \in \mathbb{N} \cup \{\infty\}$. The first term of $f_w(\theta, o_1)$ only depends on the type of agent 1, and increases in its valuation for outcome o_1, while the remaining terms ignore θ_1 entirely.

This restriction to admissible discriminant functions is crucial because it allows us to directly infer agent-independent prices from the discriminant function of a trained classifier. For this, define the *associated price function* of an admissible classifier h_w, as

$$t_w(\theta_{-1}, o_1) = -\frac{1}{w_1} w_{-1}^T \psi(\theta_{-1}, o_1),$$

where we again focus on agent 1 for concreteness. By agent symmetry, we obtain the mechanism (g, p_w) corresponding to classifier h_w, by defining payment rule,

$$p_w(\theta) = \big(t_w(\theta_{-1}, g_1(\theta)), t_w(\theta_{-2}, g_2(\theta)), \ldots, t_w(\theta_{-n}, g_n(\theta))\big).$$

Even requiring admissibility, the hope is that appropriate choices for the feature map ψ can produce rich function spaces, and thus ultimately useful payment rules. Moreover, this admissibility structure can be adopted in the context of structural support vector machines, as discussed in Section 9.5.1.

9.4.2 Example: Single-Item Auction

Before proceeding further, we illustrate the ideas developed so far in the context of a single-item auction. In a single-item auction, the type of each agent is a single number, corresponding to its value for the item, and there are two possible allocations from the point of view of an agent: one where it receives the item, and one where it does not. Formally, $\Theta = \mathbb{R}^n$ and $\Omega_1 = \{0, 1\}$ (agent 1 is allocated, or it is not).

Consider a setting with three agents and a training set:

$$(\theta^1, o_1^1) = ((1, 3, 5), 0), \quad (\theta^2, o_1^2) = ((5, 4, 3), 1), \quad (\theta^3, o_1^3) = ((2, 3, 4), 0),$$

and note that this training set is consistent with an *optimal* outcome rule, i.e., one that assigns the item to an agent with maximum value.

Our goal is to learn an admissible classifier,

$$h_w(\theta) = \arg\max_{o_1 \in \{0,1\}} f_w(\theta, o_1) = \arg\max_{o_1 \in \{0,1\}} w_1 v_1(\theta_1, o_1) + w_{-1}^T \psi(\theta_{-1}, o_1),$$

that performs well on the training set. Since there are only two possible outcomes, the outcome chosen by h_w is simply the one with the larger discriminant. A classifier that is perfect on the training data must therefore satisfy the following constraints:

$$\begin{aligned}
w_1 \cdot 0 + w_{-1}^T \psi((3,5),0) &> w_1 \cdot 1 + w_{-1}^T \psi((3,5),1),\\
w_1 \cdot 5 + w_{-1}^T \psi((4,3),1) &> w_1 \cdot 0 + w_{-1}^T \psi((4,3),0),\\
w_1 \cdot 0 + w_{-1}^T \psi((3,4),0) &> w_1 \cdot 2 + w_{-1}^T \psi((3,4),1).
\end{aligned}$$

This can for example be achieved by setting $w_1 = 1$, and

$$w_{-1}^T \psi((\theta_2, \theta_3), o_1) = \begin{cases} -\max(\theta_2, \theta_3) & \text{if } o_1 = 1 \text{ and} \\ 0 & \text{if } o_1 = 0. \end{cases} \tag{9.1}$$

Recalling our definition of the price function as $t_w(\theta_{-1}, o_1) = -(1/w_1) w_{-1}^T \psi(\theta_{-1}, o_1)$, we see that this choice of w and ψ corresponds to the second-price payment rule.

In practice, we are limited to hypotheses that are linear in features $\psi((\theta_2, \theta_3), o_1)$, and should not expect that the classifier is exact on the training data or generally on the distribution of inputs. Nevertheless, we will see in Section 9.5.1 that through the use of kernels we can adopt choices of ψ that allow for rich, non-linear discriminant functions.

9.4.3 Perfect Classifiers and Implementable Outcome Rules

We now formally establish a connection between mechanism design and multi-class classification.

Theorem 9.4.1. *Let (g, p) be a strategyproof mechanism with an agent symmetric outcome rule g, and let t_1 be the corresponding price function. Then, a perfect admissible classifier h_w for partial outcome rule g_1 exists if* $\arg\max_{o_1 \in \Omega_1} (v_1(\theta_1, o_1) - t_1(\theta_{-1}, o_1)))$ *is unique for every type profile θ.*

Proof. By the first characterization of strategyproof mechanisms, g must select an outcome

that maximizes the utility of agent 1 at the current prices, i.e.,

$$g_1(\theta) \in \arg\max_{o_1 \in \Omega_1}(v_1(\theta_i, o_1) - t_1(\theta_{-1}, o_1)).$$

Consider the admissible discriminant $f_{(1,1)}(\theta, o_1) = v_1(\theta_1, o_1) - t_1(\theta_{-1}, o_1)$, which uses the price function t_1 as its feature map. Clearly, the corresponding classifier $h_{(1,1)}$ maximizes the same quantity as g_1, and the two must agree if there is a unique maximizer. □

The relationship also works in the opposite direction: a perfect, admissible classifier h_w for outcome rule g can be used to construct a payment rule that turns g into a strategyproof mechanism.

Theorem 9.4.2. *Let g be an agent symmetric outcome rule, $h_w : \Theta \to \Omega_1$ an admissible classifier, and p_w the payment rule corresponding to h_w. If h_w is a perfect classifier for the partial outcome rule g_1, then mechanism (g, p_w) is strategyproof.*

We prove this result by expressing the regret of an agent in mechanism (g, p_w) in terms of the discriminant function f_w. Let $\Omega_i(\theta_{-i}) \subseteq \Omega_i$ denote the set of partial outcomes for agent i that can be obtained under g given reported types θ_{-i} from all agents but i, keeping the dependence on g silent for notational simplicity.

Lemma 9.4.3. *Suppose that agent 1 has type θ_1 and that the other agents report types θ_{-1}. Then the regret of agent 1 for bidding truthfully in mechanism (g, p_w) is*

$$\frac{1}{w_1}\big(\max_{o_1 \in \Omega(\theta_{-1})} f_w(\theta, o_1) - f_w(\theta, g_1(\theta))\big).$$

Proof. We have

$$\begin{aligned}
rgt_1(\theta) = & \max_{\theta_1' \in \Theta_1} \big(v_1(\theta_1, g_1(\theta_1', \theta_{-1})) - p_{w,1}(\theta_1', \theta_{-1})\big) - \big(v_1(\theta_1, g_1(\theta)) - p_{w,1}(\theta)\big) \\
= & \max_{o_1 \in \Omega_1(\theta_{-1})} \big(v_1(\theta_1, o_1) - t_w(\theta_{-1}, o_1)\big) - \big(v_1(\theta_1, g_1(\theta)) - t_w(\theta_{-1}, g_1(\theta))\big) \\
= & \max_{o_1 \in \Omega_1(\theta_{-1})} \big(v_1(\theta_1, o_1) + \frac{1}{w_1} w_{-1}^T \psi(\theta_{-1}, o_1)\big) - \big(v_1(\theta_1, g_1(\theta)) + \frac{1}{w_1} w_{-1}^T \psi(\theta_{-1}, g_1(\theta))\big) \\
= & \frac{1}{w_1}\big(\max_{o_1 \in \Omega_1(\theta_{-1})} f_w(\theta, o_1) - f_w(\theta, g_1(\theta))\big).
\end{aligned}$$

□

Proof of Theorem 9.4.2. If h_w is a perfect classifier, then the discriminant function f_w satisfies $\arg\max_{o_1 \in \Omega_1} f_w(\theta, o_1) = g_1(\theta)$ for every $\theta \in \Theta$. Since $g_1(\theta) \in \Omega_1(\theta_{-1})$, we thus have

that $\max_{o_1 \in \Omega_1(\theta_{-1})} f_w(\theta, o_1) = f_w(\theta, g_1(\theta))$. By Lemma 9.4.3, the regret of agent 1 for bidding truthfully in mechanism (g, p_w) is always zero, which means that the mechanism is strategyproof. □

It bears emphasis that classifier h_w is only used to derive the payment rule p_w, while the outcome is still selected according to g.

We might ask whether classifier h_w could be used to obtain an agent symmetric outcome rule g_w, and, since h_w is a perfect classifier for itself, a strategyproof mechanism (g_w, p_w). In particular, for each agent i, the outcome rule g_w would be defined to select the outcome o_i^* that maximizes, $f_w(\theta, o_i) = w_i v_i(\theta_i, o_i) + w_{-i}^T \psi(\theta_{-i}, o_i)$. But the problem is that this need not be feasible: there need not be a set of outcomes, $o^* = (o_1^*, \ldots, o_n^*)$, such that this outcome is itself feasible. For example, in the context of an auction, the outcome rule g_w implied by the trained classifier might seek to give the same item to the more than one agent.

The mechanism that we adopt, namely (g, p_w), has in some sense the opposite problem—it is guaranteed to be feasible because outcome rule g is feasible, but is only strategyproof if h_w is a perfect classifier for g. While the learned payment rule, p_w, always satisfies the agent-independent property (2.1), the agent-maximizing property (2.2) (the second requirement for strategyproofness) is violated when $h_w(\theta) \neq g_1(\theta)$.

9.4.4 Approximate Classification and Approximate Strategyproofness

A perfect admissible classifier for outcome rule g provides a payment rule for a strategyproof mechanism. We now show that this result extends gracefully to situations where no such payment rule is available, by relating the *expected* ex post regret of a mechanism (g, p) to a measure of the generalization error of a classifier for outcome rule g.

Fix a feature map ψ, and denote by $\mathcal{H}_\psi$ the space of all admissible classifiers with this feature map. The *discriminant loss* of a classifier $h_w \in \mathcal{H}_\psi$ with respect to a type profile θ and an outcome $o_1 \in \Omega_1$ is given by,

$$\Delta_w(o_1, \theta) = \frac{1}{w_1}\big(f_w(\theta, h_w(\theta)) - f_w(\theta, o_1)\big).$$

Intuitively the discriminant loss measures how far, in terms of the normalized discriminant, h_w is from predicting the correct outcome for type profile θ, assuming the correct outcome is o_1. Note that $\Delta(o_1, \theta) \geq 0$ for all $o_1 \in \Omega_1$ and $\theta \in \Theta$, and $\Delta(o_1, \theta) = 0$ if $o_1 = h_w(\theta)$. In addition, $h_w(\theta) = h_{w'}(\theta)$ does not imply that $\Delta_w(o_1, \theta) = \Delta_{w'}(o_1, \theta)$ for all $o_1 \in \Omega_1$: even if

two classifiers predict the same outcome, one of them may still be closer to predicting the correct outcome o_1.

The *generalization error* of classifier $h_w \in \mathcal{H}_\psi$ with respect to a type distribution D and a partial outcome rule $g_1 : \Theta \to \Omega_1$, is given by

$$R_w(D, g) = \int_{\theta \in \Theta} \Delta_w\big(g_1(\theta), \theta\big) D(\theta) d\theta.$$

The following result establishes a connection between the generalization error and the expected ex post regret of the corresponding mechanism.

Theorem 9.4.4. *Consider an outcome rule g, a space $\mathcal{H}_\psi$ of admissible classifiers, and a type distribution D. Let $h_{w^*} \in \mathcal{H}_\psi$ be a classifier that minimizes generalization error with respect to D and g among all classifiers in $\mathcal{H}_\psi$. Then the following holds:*

1. *If g satisfies consumer sovereignty, then (g, p_{w^*}) minimizes expected ex post regret with respect to D among all mechanisms (g, p_w) corresponding to classifiers $h_w \in \mathcal{H}_\psi$.*
2. *Otherwise, (g, p_{w^*}) minimizes an upper bound on expected ex post regret with respect to D amongst all mechanisms (g, p_w) corresponding to classifiers $h_w \in \mathcal{H}_\psi$.*

Proof. For the second property, observe that

$$\begin{aligned}
\Delta_w(g_1(\theta), \theta) &= \frac{1}{w_1}\big(f_w(\theta, h_w(\theta)) - f_w(\theta, g_1(\theta))\big) \\
&= \frac{1}{w_1}\big(\max_{o_1 \in \Omega_1} f_w(\theta, o_1) - f_w(\theta, g_1(\theta))\big) \\
&\geq \frac{1}{w_1}\big(\max_{o_1 \in \Omega(\theta_{-1})} f_w(\theta, o_1) - f_w(\theta, g_1(\theta))\big) = rgt_1(\theta),
\end{aligned}$$

where the last equality holds by Lemma 9.4.3. If g satisfies consumer sovereignty, then the inequality holds with equality, and the first property follows as well. □

Minimization of expected regret itself, rather than an upper bound, can also be achieved even in the absence of consumer sovereignty (which holds for all the outcome rules studied in this paper) if the learner has access to the set of available outcomes, $\Omega_1(\theta_{-1})$, that are achievable for every $\theta_{-1} \in \Theta_{-1}$.

9.5 A Solution using Structural Support Vector Machines

In this section we discuss the method of *structural support vector machines* (structural SVMs) [Tsochantaridis et al., 2005, Joachims et al., 2009]. In particular, we show how

structural SVMs can be adapted for the purpose of learning classifiers with admissible discriminant functions.

9.5.1 Structural SVMs

Given an input space X, a discrete output space Y, a target function $h^* : X \to Y$, and a set of *training examples* $\{(x^1, h^*(x^1)), \ldots, (x^\ell, h^*(x^\ell))\} = \{(x^1, y^1), \ldots, (x^\ell, y^\ell)\}$, structural SVMs learn a multi-class classifier h that given input $x \in X$ selects an output $y \in Y$ to maximize $f_w(x, y) = w^T \psi(x, y)$. For a given feature map ψ, the training problem is to find a vector w for which h_w has low generalization error.

Given examples $\{(x^1, y^1), \ldots, (x^\ell, y^\ell)\}$, training is achieved by solving the following convex optimization problem:

$$\min_{w,\xi \geq 0} \ \frac{1}{2} w^T w + \frac{C}{\ell} \sum_{k=1}^{\ell} \xi^k \qquad \text{(Training Problem 1)}$$

$$\text{s.t. } w^T \big(\psi(x^k, y^k) - \psi(x^k, y)\big) \geq \mathcal{L}(y^k, y) - \xi^k \quad \text{for all } k = 1, \ldots, \ell,\ y \in Y$$

$$\xi^k \geq 0 \quad \text{for all } k = 1, \ldots, \ell.$$

The goal is to find a weight vector w and slack variables ξ^k such that the objective function is minimized while satisfying the constraints. The learned weight vector w parameterizes the discriminant function f_w, which in turn defines the classifier h_w. The kth set of constraints state that the value of the discriminant function on (x^k, y^k) should exceed the value of the discriminant function on (x^k, y) by at least $\mathcal{L}(y^k, y)$, where $\mathcal{L}$ is a loss function that penalizes misclassification, with $\mathcal{L}(y, y) = 0$ and $\mathcal{L}(y, y') \geq 0$ for all $y, y' \in Y$. We generally use a 0/1 loss function, but consider an alternative in Section 9.5.2 to improve ex post IR properties. Positive values for the slack variables ξ^k allow the weight vector to violate some of the constraints.

The other term in the objective, the squared norm of the weights, penalizes larger weight vectors. Without this, scaling up the weight vector w can arbitrarily increase the margin between $f_w(x^k, y^k)$ and $f_w(x^k, y)$, and make the constraints easier to satisfy. Smaller values of w, on the other hand, increases the ability of the learned classifier to generalize by decreasing the propensity to over-fit to the training data.

Parameter $C \geq 0$ is a *regularization parameter*: larger values of C encourage small ξ^k and larger w, such that more points are classified correctly, but with a smaller margin (and thus perhaps with less generalization power).

The Feature Map and the Kernel Trick

Given a feature map ψ, the *feature vector* $\psi(x, y)$ for $x \in X$ and $y \in Y$ provides an alternate representation of the input-output pair (x, y). It is useful to consider feature maps ψ for which $\psi(x, y) = \phi(\chi(x, y))$, where $\chi : X \times Y \to \mathbb{R}^s$ for some $s \in \mathbb{N}$ is an *attribute map* that combines x and y into a single *attribute vector*, $\chi(x, y)$, which compactly represents the pair. Given this, function $\phi : \mathbb{R}^s \to \mathbb{R}^k$, for $b > s$, maps the attribute vector to a higher-dimensional space and can introduce additional non-linear interactions between attributes. In this way, SVMs can achieve non-linear classification in the attribute space.

What is commonly described as "feature engineering" occurs through a combination of designing a good attribute map and also defining a good function ϕ to map the attribute vector to a higher-dimensional feature vector. We insist that the size of the attribute vector s is small enough to be manageable. On the other hand, through the use of kernels we can allow for a large and even unbounded b, because in the dual of Training Problem 1, $\psi(x, y)$ only appears in an inner product of the form $\langle \psi(x, y), \psi(x', y') \rangle$, or, for a decomposable feature map, $\langle \phi(q), \phi(q') \rangle$ where $q = \chi(x, y)$ and $q' = \chi(x', y')$. For computational tractability it suffices that this inner product can be computed efficiently, and the kernel "trick" is to choose ϕ such that $\langle \phi(q), \phi(q') \rangle = K(q, q')$ for a simple closed-form function K, which is known as the *kernel*.

Two common kernels are the *polynomial kernel* K_{polyd}, which is parameterized by degree $d \in \mathbb{N}^+$, and the *radial basis function (RBF) kernel* K_{RBF}, which is parameterized by $\gamma = 1/(2\sigma^2)$ for $\sigma \in \mathbb{R}^+$:

$$K_{polyd}(q, q') = (q \cdot q')^d,$$
$$K_{RBF}(q, q') = \exp\left(-\gamma \left(\|q\|^2 + \|q'\|^2 - 2q \cdot q'\right)\right).$$

Both polynomial and RBF kernels use the standard inner product of their arguments, so their efficient computation requires only that $\chi(x, y) \cdot \chi(x, y')$ can be computed efficiently. A polynomial kernel of degree 1 is known as a linear kernel and simply scales the components of the attribute vector. In our experimental results we adopt the RBF kernel for part of our study on CAs, but develop our other experimental results without making use of the kernel trick.

Dealing with an Exponentially Large Output Space

Training Problem 1 has $\Omega(|Y|\ell)$ constraints, where Y is the output space and ℓ the number of training instances, and enumerating all of them is computationally prohibitive when Y is large. Joachims et al. [2009] address this issue for structural SVMs through constraint generation: starting from an empty set of constraints, this technique iteratively adds a constraint that is maximally violated by the current solution until that violation is below a desired threshold ϵ. Joachims et al. show that this will happen after no more than $O(\frac{C}{\epsilon})$ iterations, each of which requires $O(\ell)$ (resp. $O(\ell^2)$) time and memory if linear (resp. polynomial or RBF) kernels are used.

However, this approach assumes the existence of an *efficient separation oracle*, which given a weight vector w, an input x^k, and a target y^k, finds an output $y^* \in \arg\max_{y \in Y} f_w(x^k, y) + \mathcal{L}(y^k, y)$. The existence of such an oracle remains an open question in application to multi-minded CAs; see Section 9.6.1 for additional discussion on this.

Sometimes the subproblem $\max_{y \in Y} f_w(x^k, y) + \mathcal{L}(y^k, y)$ can be written as a polynomially sized linear program of a particular form. We will see this in the context of succinct, graphical representations of agent valuations in the CA domain. In this case, we can modify Training Problem 1 so that constraint generation is not needed, even when the output space is exponential in the problem size Taskar et al. [2004]. Indeed, suppose that we can write $\max_{y \in Y} f_w(x^k, y) + \mathcal{L}(y^k, y)$ as a linear program of the form:

$$\begin{aligned} \max \quad & wBz \\ \text{subject to} \quad & z \geq 0, Az \leq b, \end{aligned} \tag{9.2}$$

where A, B, b are functions of x^k and w is assumed to be given. Assuming that this program is feasible and bounded, we have a dual linear program that attains the same objective value:

$$\begin{aligned} \min \quad & b^T z' \\ \text{subject to} \quad & z' \geq 0, A^T z' \geq (wB)^T. \end{aligned} \tag{9.3}$$

In this case, we can rewrite Training Problem 1 by replacing the many constraints for a

single training example with a single constraint that uses a max function:

$$\begin{aligned}
\min_{w,\xi\geq 0}\ & \frac{1}{2}w^T w + \frac{C}{\ell}\sum_{k=1}^{\ell}\xi^k \\
\text{s.t.}\ & w^T\psi(x^k,y^k) + \xi^k \geq \max_{y\in Y}\Big(w^T\psi(x^k,y) + \mathcal{L}(y^k,y)\Big) \quad \text{for all } k=1,\ldots,\ell \\
& \xi^k \geq 0 \quad \text{for all } k=1,\ldots,\ell.
\end{aligned}$$

We can now apply the LP formulation for finding the maximal value of $f_w(x^k,y) + \mathcal{L}(x,y)$.

$$\begin{aligned}
\min_{w,\xi\geq 0}\ & \frac{1}{2}w^T w + \frac{C}{\ell}\sum_{k=1}^{\ell}\xi^k \\
\text{s.t.}\ & w^T\psi(x^k,y^k) + \xi^k \geq \max_{z\geq 0, A^k z\leq b^k} wB^k z \quad \text{for all } k=1,\ldots,\ell \\
& \xi^k \geq 0 \quad \text{for all } k=1,\ldots,\ell.
\end{aligned}$$

By LP duality, we can replace the max linear program with a min linear program.

$$\begin{aligned}
\min_{w,\xi\geq 0}\ & \frac{1}{2}w^T w + \frac{C}{\ell}\sum_{k=1}^{\ell}\xi^k \\
\text{s.t.}\ & w^T\psi(x^k,y^k) + \xi^k \geq \min_{z\geq 0, (A^k)^T z\geq (wB^k)^T} (b^k)^T z \quad \text{for all } k=1,\ldots,\ell \\
& \xi^k \geq 0 \quad \text{for all } k=1,\ldots,\ell.
\end{aligned}$$

We can now drop the min on the right hand side since for a fixed weight vector w, the objective tries to minimize ξ^k, so the right hand side will be minimized even if we do not explicitly require this. We therefore have a single, succinct primal convex program even though the number of original constraints was exponentially large:

$$\begin{aligned}
\min_{w,\xi\geq 0}\ & \frac{1}{2}w^T w + \frac{C}{\ell}\sum_{k=1}^{\ell}\xi^k \\
\text{s.t.}\ & w^T\psi(x^k,y^k) + \xi^k \geq (b^k)^T z^k, z^k \geq 0, (A^k)^T z^k \geq (wB^k)^T \quad \text{for all } k=1,\ldots,\ell \\
& \xi^k \geq 0 \quad \text{for all } k=1,\ldots,\ell.
\end{aligned}$$

We apply these ideas in Section 9.6.2 to combinatorial auctions where agents have succinct, graph-based value representations. This allows us to have a scalable training problem

even though the winner determination problem remains NP-hard. Though we work directly with the features $\psi(x^k, y^k)$ in our experiments, it is still possible to use kernels in conjunction with the succinct formulation of the convex program. This would require working with the dual of the succinct primal convex program (see Taskar et al. [2004] for more details).

Required Information

In summary, the use of structural SVMs requires specification of the following:

1. The input space X, the discrete output space Y, and examples of input-output pairs.
2. An attribute map $\chi : X \times Y \to \mathbb{R}^s$. This function generates an attribute vector that combines the input and output data into a single object.
3. A kernel function $K(q, q')$, typically chosen from a well-known set of candidates, e.g., polynomial or RBF. The kernel implicitly calculates the inner product $\langle \phi(q), \phi(q') \rangle$, e.g., between a mapping of the inputs into a high dimensional space.
4. If the space Y is prohibitively large, and we wish to scale our training problem to a large number of training examples, we require either:
   (a) a routine that allows for efficient separation, i.e., a polynomial time algorithm that computes $\arg\max_{y \in Y} f_w(x^k, y) + \mathcal{L}(y^k, y)$ for a given w, x, to allow for constraint generation or,
   (b) a polynomially sized LP that solves $\max_{y \in Y} f_w(x^k, y) + \mathcal{L}(y^k, y)$, to enable the formulation of the training problem as a polynomially sized primal convex optimization problem.

In addition, the user needs to stipulate particular training parameters, such as the regularization parameter C, and the kernel parameter γ if the RBF kernel is being used.

9.5.2 Structural SVMs for Mechanism Design

We now specialize structural SVMs such that the learned discriminant function will provide a payment rule, for a given symmetric outcome function g and distribution D. In this application, the input domain X is the space of type profiles Θ, and the output domain Y is the space Ω_1 of outcomes for agent 1.

We construct training data by sampling $\theta \sim D$ and applying g to these inputs:

$$\{(\theta^1, g_1(\theta^1)), \ldots, (\theta^\ell, g_1(\theta^\ell))\} = \{(\theta^1, o_1^1), \ldots, (\theta^\ell, o_1^\ell)\}.$$

For admissibility of the learned hypothesis $h_w(\theta) = \arg\max_{o_1 \in \Omega_1} w^T \psi(\theta, o_1)$, we require that

$$\psi(\theta, o_1) = (v_1(\theta_1, o_1), \psi'(\theta_{-1}, o_1))$$

For this reason, we must use an attribute map $\chi' : \Theta_{-1} \times \Omega_1 \to \mathbb{R}^s$ rather than $\chi : \Theta \times \Omega_1 \to \mathbb{R}^s$, and the kernel ϕ' we specify will only be applied to the output of χ'. Given these mappings, we let $\psi'(\theta_{-1}, o_1) = \phi'(\chi'(\theta_{-1}, o_1))$. This results in the following more specialized training problem:

$$\min_{w, \xi \geq 0} \frac{1}{2} w^T w + \frac{C}{\ell} \sum_{k=1}^{\ell} \xi^k \qquad \text{(Training Problem 2)}$$

$$\text{s.t. } (w_1 v_1(\theta_1^k, o_1^k) + w_{-1}^T \psi'(\theta_{-1}^k, o_1^k)) - (w_1 v_1(\theta_1^k, o_1) + w_{-1}^T \psi'(\theta_{-1}^k, o_1)) \geq \mathcal{L}(o_1^k, o_1) - \xi^k$$

$$\text{for all } k = 1, \ldots, \ell,\ o_1 \in \Omega_1$$

$$\xi^k \geq 0 \quad \text{for all } k = 1, \ldots, \ell.$$

If $w_1 > 0$ then the weights w together with the feature map ψ' define a price function $t_w(\theta_{-1}, o_1) = -(1/w_1) w_{-1}^T \psi'(\theta_{-1}, o_1)$ that can be used to define payments $p_w(\theta)$, as described in Section 9.4.1. In this case, we can also relate the regret in the induced mechanism (g, p_w) to the classification error as described in Section 9.4.3.

Theorem 9.5.1. *Consider training data $\{(\theta^1, o_1^1), \ldots, (\theta^\ell, o_1^\ell)\}$. Let g be an outcome function such that $g_1(\theta^k) = o_1^k$ for all k. Let w, ξ^k be the weight vector and slack variables output by Training Problem 2, with $w_1 > 0$. Consider corresponding mechanism (g, p_w). For each type profile θ^k in the training data,*

$$rgt_1(\theta^k) \leq \frac{1}{w_1} \xi^k$$

Proof. Consider input θ^k. The constraints in the training problem impose that for every outcome $o_1 \in \Omega_1$,

$$w_1 v_1(\theta_1^k, o_1^k) + w_{-1}^T \psi'(\theta_{-1}^k, o_1^k) - \left(w_1 v_1(\theta_1^k, o_1) + w_{-1}^T \psi'(\theta_{-1}^k, o_1)\right) \geq \mathcal{L}(o_1^k, o_1) - \xi^k$$

Rearranging,

$$\begin{aligned}
&\xi^k \geq \mathcal{L}(o_1^k, o_1) + \left(w_1 v_1(\theta_1^k, o_1) + w_{-1}^T \psi'(\theta_{-1}^k, o_1)\right) - \left(w_1 v_1(\theta_1^k, o_1^k) + w_{-1}^T \psi'(\theta_{-1}^k, o_1^k)\right) \\
\Rightarrow\ &\xi^k \geq \mathcal{L}(o_1^k, o_1) + f_w(\theta^k, o_1) - f_w(\theta^k, o_1^k)
\end{aligned}$$

This inequality holds for every $o_1 \in \Omega_1$, so

$$\begin{aligned}
\xi^k &\geq \max_{o_1 \in \Omega_1} \left(\mathcal{L}(o_1^k, o_1) + f_w(\theta^k, o_1) - f_w(\theta^k, o_1^k)\right) \\
&\geq \max_{o_1 \in \Omega_1} \left(f_w(\theta^k, o_1) - f_w(\theta^k, o_1^k)\right) \\
&\geq w_1 rgt_1(\theta^k),
\end{aligned}$$

where the second inequality holds because $\mathcal{L}(o_1^k, o_1) \geq 0$, and the final inequality follows from Lemma 9.4.3. This completes the proof. □

We choose not to enforce $w_1 > 0$ explicitly in Training Problem 2 as it would require a custom formulation of the dual problem. Instead, in our experiments we simply discard hypotheses where the result of training is $w_1 \leq 0$. This is sensible since the discriminant function value should increase as an agent's value increases, and negative values of w_1 typically mean that the training parameter C or the kernel parameter γ (if the RBF kernel is used) are poorly chosen.

Looking forward to our experiments, this requirement of positive w_1 did not present a practical concern. For example, for multi-minded combinatorial auctions, $1049/1080 > 97\%$ of the trials had positive w_1 for the trained classifier, and for the egalitarian assignment problem all of the trained classifiers had $w_1 > 0$. In the discussion of a tractable training formulation for positive k-wise dependent valuations, we directly impose the constraint that $w_1 = 1$ since the tractable training formulation has a succinct primal which can be solved directly. We believe the constraint can be imposed more generally, but it would require a custom dual formulation of the structural SVM training, and we leave this to future work.

Payment Normalization

One issue with the framework as stated is that the payments p_w computed from the solution to Training Problem 2 could be negative. We solve this problem by normalizing payments, using a *baseline outcome* $o_\flat$. If there exists a null outcome o', such that $v_1(\theta_1, o') = 0$ for every θ_1, then this outcome provides the baseline. For example, in CAs, the null outcome

is the empty bundle. Otherwise, we adopt as the baseline outcome the outcome o_b with the lowest price to agent 1 for a given set of types of other agents. For this, let $t_w(\theta_{-1}, o_1)$ be the price function corresponding to the solution w to Training Problem 2. Adopting the baseline outcome o_b, the *normalized payments* $t'_w(\theta_{-1}, o_1)$, are defined as

$$t'_w(\theta_{-1}, o_1) = \max(0, t_w(\theta_{-1}, o_1) - t_w(\theta_{-1}, o_b)).$$

Even when the baseline outcome is defined as that with the lowest price, it is still only a function of the types of other agents θ_{-1}, and so the prices t'_w remain a function of θ_{-1} and o_1 and are still agent independent.

Individual Rationality Violation

Even after normalization, the learned payment rule p_w may not satisfy individual rationality (IR). Recall that this requires that an agent's payment is no greater than its reported value for the outcome. We offer three solutions to this problem, which can also be used in combination.

Payment offsets One way to reduce IR violations is to make an additional adjustment to prices, across all type reports, designed to reduce the prices. In particular, for a given offset $\mathit{off} > 0$, and given normalized prices t'_w, we can then further adjust prices by the offset to obtain final prices $t''_w(\theta_{-1}, o_1) = \max(0, t'_w(\theta_{-1}, o_1) - \mathit{off})$. The effect is to leave the price on the baseline outcome unchanged (since its price was already normalized to zero), but to apply the offset where possible to other outcomes. The offset must chosen in an agent independent way, either as a fixed offset applied to all instances or an offset that depends only on θ_{-1}. In our experiments we only consider offsets that are uniformly applied to all instances.

Although the use of a payment offset decreases the IR violation it might increase regret because of the non-linearity in taking the max with zero. For instance, suppose there are only two outcomes o_{11}, o_{12}, where o_{12} is the null outcome. Suppose agent 1 values o_{11} at 5 and receives the null outcome if he reports truthfully. Suppose further that payments t_w are 7 for o_{11} and 0 for the null outcome. With no payment offset, the agent experiences no regret, since he receives utility 0 from the null outcome, but negative utility from o_{11}. However, if the payment offset is greater than 2, the agent's regret becomes positive (assuming consumer sovereignty), because he could have reported differently and received o_{11} and received positive utility.

Adjusting the loss function $\mathcal{L}$ We incur an IR violation when there is a null outcome o_{null} (for example allocating no items to an agent in a combinatorial auction), such that $g_1(\theta) \neq o_{null}$ and $f_w(\theta, o_{null}) > f_w(\theta, g_1(\theta))$ for some type θ; i.e., the discriminant value of the null outcome is greater than that for the actual outcome selected by the outcome rule. This happens because the discriminant $f_w(\theta, o_1)$ is a scaled version of the agent's utility for outcome o_1 under payments p_w. If the utility for the null outcome is greater than the utility for $g_1(\theta)$, and the payment on null outcomes is normalized to zero, then the payment $t_w(\theta_{-1}, g_1(\theta))$ must be greater than $v_1(\theta_1, g_1(\theta))$ (so that the discriminant value $f_w(\theta, g_1(\theta)) < f_w(\theta, o_{null})$), causing an IR violation.

Recognizing this, we can discourage these types of errors by modifying the constraints of Training Problem 2: when $o_1^k \neq o_{null}$ and $o_1 = o_{null}$, we can increase $\mathcal{L}(o_1^k, o_1)$ to heavily penalize misclassifications of this type. With a larger $\mathcal{L}(o_1^k, o_1)$, a larger ξ^k will be required if $f_w(\theta, o_1^k) < f_w(\theta, o_{null})$. As with payment offsets, this technique will decrease IR violations but is not guaranteed to eliminate all of them. In our experimental results, we refer to this as the *null loss fix*, and the null loss refers to the value we choose for $\mathcal{L}(o_1^k, o_{null})$ where outcome $o_1^k \neq o_{null}$.

Deallocation In settings that have a null outcome and are *downward closed* (i.e., settings where a feasible outcome o remains feasible if o_i is replaced with the null outcome), we can also choose to modify the function g to allocate the null outcome whenever the price function t_w creates an IR violation. This reduces ex post regret, and in particular ensures ex post IR for all instances. On the other hand, the total value to the agents necessarily decreases under the modified allocation, and we begin to deviate from the intended outcome rule. In our experimental results, we refer to this as the *deallocation fix*.

9.6 Applying the Framework

In this section, we discuss the application of our framework to three domains: multi-minded combinatorial auctions, combinatorial auctions with k-wise dependent valuations, and egalitarian assignment.

9.6.1 Multi-Minded Combinatorial Auctions

We adopt the same notation introduced in Section 7.1. Recall that we have a set N of agents and a set $G = \{1, \ldots, m\}$ of items, with $|N| = n$, $|G| = m$.

The outcome space Ω_i for agent i is the set of all subsets of the m items, and in full generality, the type of agent i can be represented by a vector $\theta_i \in \Theta_i = \mathbb{R}^{2^m}$ that specifies its value for each possible bundle. The set of possible type profiles is then $\Theta = \mathbb{R}^{2^m n}$, and the value $v_i(\theta_i, o_i)$ of agent i for bundle o_i is equal to the entry in θ_i corresponding to o_i.

We require that valuations are monotone, such that $v_i(\theta_i, o_i) \geq v_i(\theta_i, o_i')$ for all $o_i, o_i' \in \Omega_i$ with $o_i' \subseteq o_i$, and normalized such that $v_i(\theta_i, \emptyset) = 0$. Assuming agent symmetry and adopting the view of agent 1, the partial outcome rule $g_1 : \Theta \to \Omega_1$ specifies the bundle $g_1(\theta)$ allocated to agent 1. We require feasibility of outcome rules, so that no item is allocated more than once.

In a multi-minded CA, each agent is interested in at most κ bundles for some constant κ. The special case where $\kappa = 1$ is the well studied problem of single-minded CAs discussed in Section 7.4. If a bundle contains multiple bundles of interest, the agent's value for that bundle is the bundle with the highest value among the contained bundles of interest (i.e., we adopt XOR semantics). We choose to study multi-minded CAs rather than single-minded CAs because they provide an example for which truthful, algorithmic mechanism design is not well understood. Multi-minded CAs are an example of a multi-parameter mechanism design problem where the valuation profiles and thus the training data can still be represented in a compact way. In the case of multi-minded CAs, the compact representation arises by explicitly writing down the identities and values of an agent's κ target bundles. In addition, in multi-minded CAs, the inner products between valuation profiles, which are required to apply the kernel trick, can be computed in polynomial time.

Attribute Maps

To apply structural SVMs to multi-minded CAs, we need to specify an appropriate attribute map χ. In our experiments we use two attribute maps $\chi_1 : \Theta_{-1} \times \Omega_1 \to \mathbb{R}^{2^m(2^m(n-1))}$ and $\chi_2 : \Theta_{-1} \times \Omega_1 \to \mathbb{R}^{2^m(n-1)}$, which are defined as follows:

$$\chi_1(\theta_{-1}, o_1) = \begin{bmatrix} 0 \\ \dots \\ 0 \\ \theta_{-1} \\ 0 \\ \dots \\ 0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} 0 \\ \dots \\ 0 \end{matrix}}\right\} dec(o_1)(2^m(n-1)) \\ \\ \left.\vphantom{\begin{matrix} 0 \\ \dots \\ 0 \end{matrix}}\right\} (2^m - dec(o_1) - 1)(2^m(n-1)) \end{matrix}, \quad \chi_2(\theta_{-1}, o_1) = \begin{bmatrix} \theta_2 \setminus o_1 \\ \theta_3 \setminus o_1 \\ \dots \\ \theta_n \setminus o_1 \end{bmatrix}.$$

Here, $dec(o_1) = \sum_{j=1}^{m} 2^{j-1}\mathbf{I}_{j\in o_1}$ is a decimal index of bundle o_1, where $\mathbf{I}_{j\in o_1} = 1$ if $j \in o_1$ and $\mathbf{I}_{j\in o_1} = 0$ otherwise. Attribute map χ_1 thus stacks the vector θ_{-1}, which represents the valuations of all agents except agent 1, with zero vectors of the same dimension, where the position of θ_{-1} is determined by the index of bundle o_1. The resulting attribute vector is simple but potentially restrictive. For example, it precludes two instances with different allocated bundles from sharing attributes, which provides an obstacle to generalization of the discriminant function across bundles.

Attribute map χ_2 stacks vectors $\theta_i \setminus o_1$, which are obtained from θ_i by setting the entries for all bundles that intersect with o_1 to 0. This captures the fact that agent i cannot be allocated any of the bundles that intersect with o_1 if o_1 is allocated to agent 1. Both χ_1 and χ_2 are defined for a particular number of items and agents, and in our experiments we train a different classifier for each number of agents and items. In practice, one can pad out items and agents by setting bids to zero and train a single classifier.

Efficient Computation of Inner Products

Efficient computation of inner products is possible for both χ_1, χ_2. For both χ_1 and χ_2, computing inner products reduces to the question of whether inner products between valuation profiles are efficiently computable. For χ_1, we have that

$$\left\langle \chi_1(\theta_{-1}, o_1), \chi_1(\theta'_{-1}, o'_1) \right\rangle = \mathbf{I}_{o_1=o'_1} \sum_{i=2}^{n} \left\langle \theta_i, \theta'_i \right\rangle,$$

where indicator $\mathbf{I}_{o_1=o'_1} = 1$ if $o_1 = o'_1$ and $\mathbf{I}_{o_1=o'_1} = 0$ otherwise. For χ_2,

$$\left\langle \chi_2(\theta_{-1}, o_1), \chi_2(\theta'_{-1}, o'_1) \right\rangle = \sum_{i=2}^{n} \left\langle \theta_i \setminus o_1, \theta'_i \setminus o_1 \right\rangle.$$

We next develop efficient methods for computing the inner products $\langle \theta_i, \theta'_i \rangle$ on compactly represented valuation functions. The computation of $\langle \theta_i \setminus o_1, \theta'_i \setminus o_1 \rangle$ can be done through similar methods.

In the single-minded setting, let θ_i correspond to a bundle $S_i \subseteq \{1, \ldots, r\}$ of items with value v_i, and θ'_i correspond to a set $S'_i \subseteq \{1, \ldots, r\}$ of items valued at v'_i.

Each set containing both S_i and S'_i contributes $v_i v'_i$ to $\theta_i^T \theta'_i$, while all other sets contribute 0. Since there are exactly $2^{r-|S_i \cup S'_i|}$ sets containing both S_i and S'_i, we have

$$\theta_i^T \theta'_i = v_i v'_i 2^{r-|S_i \cup S'_i|}.$$

This is a special case of the formula for the multi-minded case.

Lemma 9.6.1. *Consider a multi-minded CA and two bid vectors x_1 and x'_1 corresponding to sets $S = \{S_1, \ldots, S_s\}$ and $S' = \{S'_1, \ldots, S'_t\}$, with associated values $v_1, \ldots, v_s$ and $v'_1, \ldots, v'_t$. Then,*

$$x_1^T x'_1 = \sum_{T \subseteq S, T' \subseteq S'} \Big((-1)^{|T|+|T'|} \cdot (\min_{S_i \in T} v_i) \cdot (\min_{S'_j \in T'} v'_j) \cdot 2^{r - |(\bigcup_{S_i \in T} S_i) \cup (\bigcup_{S'_j \in T'} S'_j)|} \Big). \quad (9.4)$$

Proof. The contribution of a particular bundle B' of items to the inner product is $(\max_{S_i \in S, S_i \subseteq B'} v_i) \cdot (\max_{S'_j \in S', S'_j \subseteq B'} v'_j)$, and thus

$$x_1^T x'_1 = \sum_{B'} \Big((\max_{\substack{S_i \in S \\ S_i \subseteq B'}} v_i) \cdot (\max_{\substack{S'_j \in S' \\ S'_j \subseteq B'}} v'_j) \Big).$$

By the maximum-minimums identity, which asserts that for any set $\{x_1, \ldots, x_n\}$ of n numbers, $\max\{x_1, \ldots, x_n\} = \sum_{Z \subseteq X} ((-1)^{|Z|+1} \cdot (\min_{x_i \in Z} x_i))$,

$$\max_{\substack{S_i \in S \\ S_i \subseteq B'}} v_i = \sum_{\substack{T \subseteq S \\ \bigcup_{S_i \in T} S_i \subseteq B'}} \Big((-1)^{|T|+1} \cdot (\min_{S_i \in T} v_i) \Big) \quad \text{and}$$

$$\max_{\substack{S'_j \in S' \\ S'_j \subseteq B'}} v'_j = \sum_{\substack{T' \subseteq S' \\ \bigcup_{S'_j \in T'} S'_j \subseteq B'}} \Big((-1)^{|T'|+1} \cdot (\min_{S'_j \in T'} v'_j) \Big).$$

The inner product can thus be written as

$$\theta_1^T \theta'_1 = \sum_{B'} \sum_{\substack{T \subseteq S, T' \subseteq S' \\ \bigcup_{S_i \in T} S_i \subseteq B' \\ \bigcup_{S'_j \in T'} S'_j \subseteq B'}} \Big((-1)^{|T|+|T'|} \cdot (\min_{S_i \in T} v_i) \cdot (\min_{S'_j \in T'} v'_j) \Big).$$

Finally, for given $T \subseteq S$ and $T' \subseteq S'$, there exist exactly $2^{r - |(\bigcup_{S_i \in T} S_i) \cup (\bigcup_{S'_j \in T'} S'_j)|}$ bundles B' such that $\bigcup_{S_i \in T} S_i \subseteq B'$ and $\bigcup_{S'_j \in T'} S'_j \subseteq B'$, and we obtain

$$\theta_1^T \theta'_1 = \sum_{T \subseteq S, T' \subseteq S'} \Big((-1)^{|T|+|T'|} \cdot (\min_{S_i \in T} v_i) \cdot (\min_{S'_j \in T'} v'_j) \cdot 2^{m - |(\bigcup_{S_i \in T} S_i) \cup (\bigcup_{S'_j \in T'} S'_j)|} \Big). \quad \square$$

$\square$

If S and S' have constant size, then the sum on the right hand side of (9.4) ranges over

a constant number of sets and can be computed efficiently.

Dealing with an Exponentially Large Output Space

Recall that Training Problems 1 and 2 have constraints for every training example (θ^k, o_1^k) and every possible bundle of items $o_1 \in \Omega_1$. For CAs, there will be exponentially many such bundles. In lieu of an efficient separation oracle, a workaround exists when the discriminant function ensures that the induced prices weakly increase as items are added to a bundle. Given this property of *item monotonicity*, it suffices to include constraints for bundles that have a strictly larger value to the agent than any of their respective subsets. Coupled with the assumption that valuations in CAs are monotone, and the admissibility property of the discriminant function, no other bundles can have a greater discriminant value than these bundles.

But imposing item monotonicity directly on the training problem requires a number of constraints that is exponential in the number of items. For polynomial kernels and certain attribute maps, a possible sufficient condition for item monotonicity is to force the weights w_{-1} to be negative. However, as with the discussion of enforcing $w_1 > 0$ directly, these weight constraints do not dualize conveniently and results in the dual formulation no longer operate on inner products $\langle \psi'(\theta_{-1}, o_1), \psi'(\theta'_{-1}, o'_1) \rangle$. As a result, we would be forced to work in the primal, and incur extra computational overhead that increases polynomially with the kernel degree d. We have performed some preliminary experiments with polynomial kernels, but we have not looked into reformulating the primal to enforce item monotonicity.

For this reason, the baseline experimental results in Section 9.7 do not assume item monotonicity, and instead use an inefficient separation oracle, that simply iterates over all possible bundles $o_1 \in \Omega_1$.

An alternative that we have also studied is to optimistically assume item monotonicity, and only include the constraints associated with bundles that are explicit in agent valuations. We also present experimental results that test this optimistic approach, and while there is a degradation in performance, results are mostly comparable. This provides a useful approach to scaling up training for representation languages such as the XOR representation adopted for multi-minded CAs for which it is simple to identify the small set of bundles that are candidates for maximizing the discriminant function (= agent utility.)

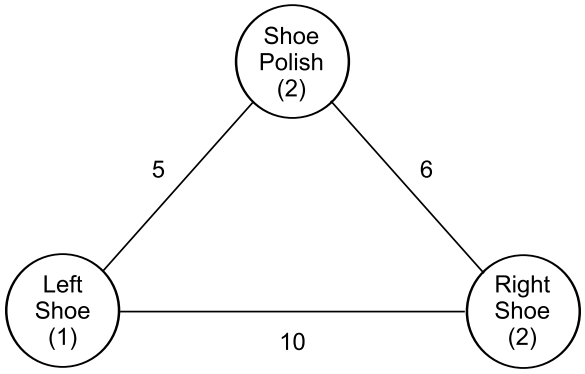


Figure 9.1: An example of a 2-wise dependent valuation. The values listed in the nodes give the agent's weights for the corresponding items. Each item has some small value on its own, but complementarities exist between pairs of items which give added utility to the agent. Note that while this graph is complete, this is not necessary. Absent edges are assumed to have weight 0.

9.6.2 Combinatorial Auctions with Positive k-wise Dependent Valuations

We also study combinatorial auctions where agents have positive k-wise dependent valuations Conitzer et al. [2005]. This setting allows us to apply the ideas discussed in Section 9.5.1 to attain a polynomial time training formulation despite the exponential size of Ω_1.

When an agent has a k-wise dependent valuation, the agent's valuation is described by a hypergraph $G = (V, E)$ with hyperedges of size at most k. The nodes in the graph correspond to the items being auctioned, and the hyperedges to groups of these items. These nodes and hyperedges are each assigned weights $g(v)$ and $g(e)$ respectively. An agent's value for a subset of items $o_1 \in \Omega_1$ is the sum of the weights of nodes and hyperedges contained in o_1, i.e., $\sum_{v \in V, v \in o_1} g(v) + \sum_{e \in E, e \subseteq o_1} g(e)$. Figure 9.1 gives a pictorial view of a simple 2-wise valuation over 3 items.

A *positive k-wise dependent valuation* adds the restriction that hyperedge weights are positive. This restriction is required for our results, and is also studied by Abraham et al. [2012]. This forces agent valuations to be superadditive, i.e. $\theta_i(o_1) \geq \theta_i(o_2) + \theta_i(o_3)$ for $o_1 = o_2 \cup o_3$, and $o_2 \cap o_3 = \emptyset$. When we have multiple agents, we use $g_i(v)$ and $g_i(e)$ to denote the weights that agent i assigns to nodes v and edges e. For convenience, let $g_i(e)$ for an edge not in the agent's edge set is defined to be 0. If we are given the agent's type θ_i, then it can be convenient to write $g(\theta_i, v)$ or $g(\theta_i, e)$ to represent the weights in the agent's underlying graph when its type is θ_i.

Though these valuations are very different from the multi-minded valuations we discussed earlier, the winner determination problem for positive k-wise dependent valuations

is still NP-hard for any value of $k > 1$ Abraham et al. [2012]. Because the winner determination problem is NP-hard, we seek to learn a payment rule for a greedy allocation algorithm.

Going forward, we specialize to the case of $k = 2$, which represents the case where the agent's hypergraph is just a graph. We believe a similar approach can work for any value of k, but we leave this to future work. The single agent winner determination problem (where nodes can have negative values but hyperedge weights are non-negative) is tractable for any k. The only complication is in designing an appropriate attribute map. Interestingly, in this case the training problem will look like a single agent winner determination problem. Intuitively, the attribute values on which the discriminant function is evaluated will model the value to the rest of the agents given that agent 1 receives a particular bundle, and this 'two-agent' view (agent 1 and the rest of the agents) can be encoded with a single valuation function and thus effectively appears as a single agent problem. This single agent problem is non-trivial as nodes can have negative values when the impact on other agents is incorporated, so the bundle of all items is not necessarily the value-maximizing subset. This single agent winner determination problem where nodes can have negative values but hyperedges have non-negative weights turns out to be tractable for k-wise dependent valuations and a suitably defined attribute map.

A Concrete Example

To clarify the construction, we introduce a simple example where agents have 2-wise dependent valuations. We refer back to this example to illustrate our greedy algorithm and attribute map. Consider a setting where we have 3 agents and 3 items. We denote the agents and items using indices $1, 2, 3$ but the association should be clear from context. The agents have the following 2-wise dependent valuations:

$$g_1(1) = 1, g_1(2) = 4, g_1(3) = 2, g_1((1,2)) = 4$$
$$g_2(1) = 2, g_2(2) = 6, g_2(3) = 2, g_2((2,3)) = 3, g_2((1,3)) = 6$$
$$g_3(1) = 5, g_3(2) = 3, g_3(3) = 1, g_3((1,2)) = 2, g_3((1,3)) = 7$$

A Greedy Algorithm

Before describing our attribute map, it will be useful to introduce a simple greedy algorithm GREEDY-KWISE that tries to find an allocation with good welfare. We use GREEDY-KWISE both in our attribute map and as an outcome rule in our experimental results.

Let $G = \{1, \ldots, m\}$ denote the set of all items. Given some subset of items $S \subseteq G$, the greedy algorithm orders the items by index and assigns the items incrementally. At each step, the algorithm computes the gain in welfare of assigning the item to each agent and chooses the agent that provides the maximal gain in welfare. Note that if an item j has been assigned to an agent i, then when considering the assignment for item k the gain in welfare of assigning it to agent i includes agent i's node weight for item k as well as agent i's edge weight for edge (j, k) (if the edge exists in the agent's valuation graph). We let $\mathsf{GREEDY\text{-}KWISE}_i(S)$ denote agent i's allocation when this greedy algorithm is run on S.

Applied to the example from Section 9.6.2, the greedy algorithm first considers the assignment of item 1. Agent 3 has the highest value, so 1 goes to agent 3. We then consider item 2. The gain in giving this to agent 1 is 4, the gain to agent 2 is 6, and the gain to agent 3 is $3 + 2 = 5$ (for agent 3, we add in both $g_3(2)$ and $g_3((1, 2))$ since 1 was given to agent 3). As a result, agent 2 has the highest gain and we give the item to agent 2. Then for item 3, the gains are 2, $2 + 3 = 5$, and $1 + 7 = 8$ respectively. As a result, item 3 is assigned to agent 3.

Attribute Map

In order to have a tractable training problem, we want our attribute map $\chi_3(\theta_{-1}, o_1)$ to be decomposable across items and pairs of items. We detail the reasons for this in the next section, but the intuition is that we want $\chi_3(\theta_{-1}, o_1)$ to resemble a 2-wise dependent valuation so that we can view the separation problem as a single agent winner determination problem for an agent with 2-wise dependent valuations where nodes can have negative weight but edges have positive weight. If nodes and edges all have positive weight, then the single agent problem has a trivial solution: take all the items. However, when nodes can have negative weight, the problem is non-trivial (even when edges weights remain positive-restricted).

Our attribute map $\chi_3(\theta_{-1}, o_1)$ maps from $\Theta_{-1} \times \Omega_1 \to \mathbb{R}^{2m+m(m-1)}$. For each possible item $j \in \{1, \ldots, m\}$, we have two entries in $\chi_3(\theta_{-1}, o_1)$.

$$V_j(0) \cdot \mathbb{I}(j \notin o_1), \quad V_j(1) \cdot \mathbb{I}(j \in o_1),$$

where $\mathbb{I}$ is an indicator variable. $V_j(0)$ approximates the "gain" of not allocating item j to agent 1; $V_j(1)$ does the opposite, approximating the "cost" of allocating item j to agent 1. We detail how these are calculated below.

Additionally, for each possible pair of items j_1, j_2, we have two entries in $\chi_3(\theta_{-1}, o_1)$:

$$V_{j_1,j_2}(0) \cdot \mathbb{I}(\{j_1, j_2\} \cap o_1 = \emptyset), \quad V_{j_1,j_2}(1) \cdot \mathbb{I}(\{j_1, j_2\} \subseteq o_1).$$

where $\mathbb{I}$ is an indicator variable, and where the $V_{j_1,j_2}(\cdot)$ indicate the value of agent one obtaining (or not) both items. To specify $V_j(0), V_j(1)$ and $V_{j_1,j_2}(0), V_{j_1,j_2}(1)$, we use GREEDY-KWISE.

We define $V_j(0)$ as the "gain" for item j by the agent who is allocated it under the greedy algorithm, i.e. $g_i(j)$ where $j \in \text{GREEDY-KWISE}_i(R)$. By contrast, we want $V_j(1)$ to be the"cost" of allocating item j to agent 1; therefore we define it as $\text{welfare}(\text{GREEDY-KWISE}(G)) - \text{welfare}(\text{GREEDY-KWISE}(G \setminus \{j\})))$. The values $V_{j_1,j_2}(0) = V_{j_1,j_2}(1)$ are similar to $V_j(0)$. These expressions look at the allocation of GREEDY-KWISE on all items, and see if j_1, j_2 are assigned to the same agent. If they are not, then they are set to zero. Otherwise, they are set to $-g((j_1, j_2))$. The negative sign here is important for tractability and ensures that edge weights are non-negative for the modified single agent problem (see the next section). The intuition for why we do not make $V_{j_1,j_2}(1)$ equal to the "cost" of allocating items j_1, j_2 is that if $j_1 \in o_1, j_2 \in o_1$ then this cost is already accounted for in $V_{j_1}(1)$ and $V_{j_2}(1)$. In fact, the cost is double-counted since in both $\text{GREEDY-KWISE}(R \setminus \{j_1\})$ and $\text{GREEDY-KWISE}(R \setminus \{j_2\})$ no agents can derive value from edge (j_1, j_2) since one of the items is missing in both cases. We use a particularly simple allocation algorithm here, but any algorithm as long as its computation time is not prohibitive.

Returning to our example from Section 9.6.2, recall that when run on all agents, the greedy algorithm gives items 1 and 3 to agent 3 and item 2 to agent 2. The total welfare in this case is 19. The total value to agents 2 and 3 is also 19 since agent 1 does not receive any items. In this case, we then have the following values for V.

- $V_1(0)$: Agent 3 receives item 1, so this is set to $g_3(1) = 5$.
- $V_2(0)$: Agent 2 receives item 2, so this is set to $g_2(2) = 6$.
- $V_3(0)$: Agent 3 receives item 3, so this is set to $g_3(3) = 1$.
- $V_1(1)$: We consider the greedy allocation where item 1 cannot be allocated. The greedy algorithm gives item 2 to agent 2, and then item 3 to agent 2 as well (since the gain will be 5 versus 1). As a result, the total welfare is 11. The welfare difference for the other agents is 8, so $V_1(1) = 8$.

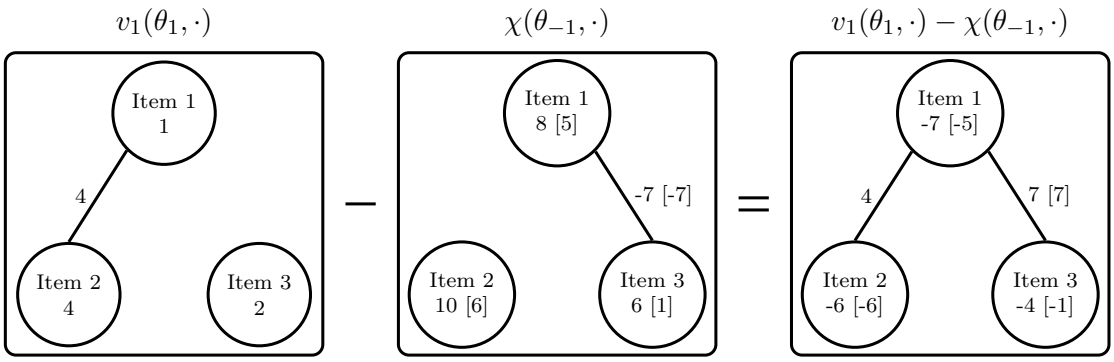


Figure 9.2: A pictorial representation of the attribute map χ_3 for our concrete example. We make the attribute map χ_3 resemble a 2-wise dependent valuation (with values for not being assigned a node and not being assigned any of the items for an edge shown in brackets) so that when combined with agent 1's valuation, we have a modified single agent problem. We do not show the weight vector w in these panels, but there would be weights $w_j(1)$ multiplying the unbracketed values in each node j, weights $w_j(0)$ multiplying the bracketed values in each node j, as well as weights $w_{j_1,j_2}(0)$ and $w_{j_1,j_2}(1)$ multiplying the unbracketed and bracketed values on the edges.

- $V_2(1)$: Without item 2, the greedy algorithm gives item 1 to agent 3 and then item 3 to agent 3 as well (gain of 8 for agent 3 versus gain of 2 for agent 1). The total welfare is 13, so $V_2(1) = 19 - 13 = 6$.
- $V_3(1)$: Without item 3, item 1 goes to agent 3 and item 2 goes to agent 2. The total welfare is 11, so $V_3(1) = 19 - 11 = 8$.
- $V_{1,2}(0), V_{1,2}(1)$: The original allocation allocates items 1 and 2 to different agents, so these values are 0.
- $V_{1,3}(0), V_{1,3}(1)$: Items 1 and 3 are allocated to agent 3, so this is set to $-g_3((1,3)) = -7$.
- $V_{2,3}(0), V_{2,3}(1)$: The original allocation allocates items 2 and 3 to different agents, so these values are 0.

A useful way to think of this attribute map χ_3 is that it modifies agent 1's 2-wise valuation. Pictorially, we can think of this as combining agent 1's valuation graph with the valuation graph induced by the feature map. See Figure 9.2 for an illustration.

A Tractable Training Problem

We are now ready to show that we can use the techniques discussed in Section 9.5.1 to make training polynomial-time in the number of items despite having an exponentially large space

of possible labels (bundles in this context).

If we assume that agents have positive 2-wise dependent valuations and we use the attribute map specified above, then we will be able to compute $\max_{o_1 \in \Omega_1} f_w(\theta, o_1) + \mathcal{L}(o_1^k, o_1)$ using a linear program whose coefficients are linear in w. For this result, we require two restrictions:

1. We impose constraints (such as positivity) on certain elements of the vector w. This restriction of the space of possible weights enables us to obtain a polynomial-time training formulation, at the possible loss of some pricing accuracy. It also prevents us from using the kernel trick over the positive-restricted weights, although we can still use kernels on the rest. In the present analysis we choose not to add this complexity and work with a linear kernel only.

2. The loss function $\mathcal{L}(o_1^k, o_1)$ is equal to 0 everywhere. This assumption simplifies our proofs, but it turns out we only require that the loss function can be expressed as a sum of products where each product consists of a weight multiplied by a.) an indicator of whether o_1 contains a given subset of items or b.) an indicator of whether o_1 does not intersect a given subset of items. As a result, we can adjust null loss by using an indicator for o_1 not intersecting the entire set of items.

In addition to these restrictions, the tractable training problem for positive k-wise dependent valuations departs from the general framework by fixing the weight w_1 to equal 1 (instead of letting it be learned by the framework, checking positivity, and dividing all terms by w_1). We believe it is possible to adopt the same constraint more generally, but it is easier in the setting for positive k-wise dependent valuations since there is a succinct primal formulation and it is not necessary to worry about modifications to the dual of the structural SVM formulation.

The proof of this result relies on a connection between k-wise dependent valuations and Markov networks, and applying a result from the literature on finding the maximum *a posteriori* (MAP) assignment in Markov networks.

Theorem 9.6.2. *When agents have positive 2-wise dependent valuations and we use the attribute map χ_3 (described above) without a kernel, then we can solve the structural SVM training problem (with the modifications discussed above in Assumptions 1 and 2.) in time polynomial in m, the number of items in the auction and n, the number of agents.*

Proof. We observe that $\chi_3(\theta_{-1}, o_1)$ is a vector with $2m + m(m-1)$ elements. Therefore, the weight vector w_{-1} will have the same number of elements. We index elements of these

vectors using notation similar to the notation we use for $\chi_3(\theta_{-1}, o_1)$. That is, we let $w_j(p)$ correspond to the attribute term that includes $V_j(p)$, where $p \in \{0, 1\}$. Similarly, we let $w_{j_1,j_2}(p)$ correspond to the attribute term that includes $V_{j_1,j_2}(p)$, where $p \in \{0, 1\}$.

In the primal formulation of Training Problem 1, we add the constraints that $w_{j_1,j_2}(p) \geq 0$ for $p \in \{0, 1\}$ and all j_1, j_2. While not strictly necessary, we also impose that $w_1 = 1$ (as we are working with the primal formulation, the enforcement of such a constraint is available to us; alternatively, we could forgo this constraint and operate in the dual, enabling the use of kernels over the unconstrained components of the attribute map).

$$
\begin{aligned}
\max_{o_1 \in \Omega_1} f_w(\theta, o_1) &= \max_{o_1 \in \Omega_1} v(\theta_1, o_1) - w_{-1}^T \chi_3(\theta_{-1}, o_1) \\
&= \max_{o_1 \in \Omega_1} \sum_{j \in G} g(\theta_1, j) \mathbb{I}(j \in o_1) + \sum_{1 \leq j_1 < j_2 \leq r} g(\theta_1, (j_1, j_2)) \mathbb{I}(\{j_1, j_2\} \subseteq o_1) \\
&\quad - \sum_{j \in G} (w_j(0) V_j(0) \mathbb{I}(j \notin o_1) + w_j(1) V_j(1) \mathbb{I}(j \in o_1)) \\
&\quad - \sum_{1 \leq j_1 < j2 \leq r} (w_{j_1,j_2}(0) V_{j_1,j_2}(0) \mathbb{I}(\{j_1, j_2\} \cap o_1 = \emptyset) \\
&\qquad\qquad + w_{j_1,j_2}(1) V_{j_1,j_2}(1) \mathbb{I}(\{j_1, j_2\} \subseteq o_1)) \\
&= \max_{o_1 \in \Omega_1} \sum_{j \in G} -w_j(0) V_j(0) \mathbb{I}(j \notin o_1) + \sum_{j \in G} (g(\theta_1, j) - w_j(1) V_j(1)) \mathbb{I}(j \in o_1) \\
&\quad + \sum_{1 \leq j_1 < j_2 \leq r} (-w_{j_1,j_2}) V_{j_1,j_2}(0) \mathbb{I}(\{j_1, j_2\} \cap o_1 = \emptyset) \\
&\quad + \sum_{1 \leq j_1 < j_2 \leq r} (g(\theta_1, (j_1, j_2)) - w_{j_1,j_2} V_{j_1,j_2}(1)) \mathbb{I}(\{j_1, j_2\} \subseteq o_1)
\end{aligned}
$$

An important observation is that the coefficients of the indicator variables $\mathbb{I}(\{j_1, j_2\} \subseteq o_1)$ and $\mathbb{I}(\{j_1, j_2) \cap o_1 = \emptyset)$ for the edges (j_1, j_2) will be positive. Indeed, as we defined in Section 9.6.2, $V_{j_1,j_2}(0) = V_{j_1,j_2}(1) \leq 0$. Combining this with the assumption that $g(\theta_1, (j_1, j_2)) \geq 0$ and our constraint that $w_{j_1,j_2} \geq 0$, we see that the coefficients of $\mathbb{I}(\{j_1, j_2\} \subseteq o_1)$ and $\mathbb{I}(\{j_1, j_2) \cap o_1 = \emptyset)$ are positive.

Applying the ideas of Taskar et al. [2004], we see that the above maximization problem can be solved by the following integer program. The integer program has a binary variable

corresponding to each of the indicator variables in the above maximization.

$$\begin{aligned}
\max \quad & \sum_{j \in G} -w_j(0)V_j(0)I_{j,0} + \sum_{j \in G}(g(\theta_1, j) - w_j(1)V_j(1))I_{j,1} \\
& + \sum_{1 \le j_1 < j_2 \le m} (-w_{j_1,j_2}(0)V_{j_1,j_2}(0))I_{j_1,j_2,0} \\
& + \sum_{1 \le j_1 < j_2 \le m} (g(\theta_1, (j_1, j_2)) - w_{j_1,j_2}(1))I_{j_1,j_2,1} \\
\text{s.t.} \quad & I_{j,0} + I_{j,1} = 1 \quad \text{for all } j \in G \\
& I_{j_1,j_2,p} \le I_{j_1,p}, I_{j_1,j_2,p} \le I_{j_2,p} \quad \text{for all } 1 \le j_1 < j_2 \le m, p \in \{0,1\} \\
& I_{j,p} \in \{0,1\} \quad \text{for all } j \in G, p \in \{0,1\} \\
& I_{j_1,j_2,p} \in \{0,1\} \quad \text{for all } 1 \le j_1 < j_2 \le m, p \in \{0,1\}
\end{aligned}$$

The first set of constraints ensures that exactly one of $I_{j,0}$ and $I_{j,1}$ is active. The second set of constraints ensures that $I_{j_1,j_2,p}$ is active if and only if $I_{j_1,p}$ and $I_{j_2,p}$ are active. Note that the 'if' direction follows because the objective coefficients of $I_{j_1,j_2,p}$ are non-negative, so the second set of constraints will be tight if $I_{j_1,p}$ and $I_{j_2,p}$ are both set to 1. Therefore, the value of the objective corresponds to $f_w(\theta, o)$, where o consists of the items j for which $I_{j,1}$ is set to one.

To complete the proof, we apply Theorem 3.1 from Taskar et al. [2004] to show that the LP relaxation of this integer program is integral. □

9.6.3 The Assignment Problem

In the assignment problem, we are given a set of n agents and a set $\{1, \ldots, n\}$ of items, and wish to assign each item to exactly one agent. The outcome space of agent i is thus $\Omega_i = \{1, \ldots, n\}$, and its type can be represented by a vector $\theta_i \in \Theta_i = \mathbb{R}^n$. The set of possible type profiles is then $\Theta = \mathbb{R}^{n^2}$.

We consider an outcome rule that maximizes *egalitarian welfare* in a lexicographic manner: first, the minimum value of any agent is maximized; if more than one outcome achieves the minimum, the second lowest value is maximized, and so forth. A simple example shows that this outcome rule violates weak monotonicity (Section 2.2.1), a necessary condition for the existence of a strategyproof payment rule.

Proposition 3. The outcome rule that maximizes egalitarian welfare for the assignment problem is not weakly monotone.

Proof. Consider a setting with two agents and two items. Denote the items by a and b. Suppose that $v_2(\theta_2, a) = 3, v_2(\theta_2, b) = 4$.

Suppose that $v_1(\theta_1, a) = 1, v_1(\theta_1, b) = 2$. When given θ_1, θ_2, the assignment that optimizes egalitarian welfare gives b to agent 1 and a to agent 2. The minimum value received by any agent is 2 (the value agent 1 receives for b).

Now consider θ_1', where $v_1(\theta_1', a) = 4, v_1(\theta_1', b) = 6$. When passed θ_1', θ_2, the assignment that optimizes egalitarian welfare gives a to agent 1 and b to agent 2. The minimum value received by any agent is 4 (the opposite assignment gives minimum value of 3).

We have that $v_1(\theta_1', a) - v_1(\theta_1', b) = -2 < -1 = v_1(\theta_1, a) - v_1(\theta_1, b)$ contradicting weak monotonicity. □

This outcome rule can be computed by solving a sequence of integer programs. As such, our focus in this application is not on studying our framework for the setting of tractable outcome rules, but rather for understanding its performance on an objective that is very different than welfare maximization. We continue to assume agent symmetry, and adopt the view of agent 1.

To complete our specification of the structural SVM framework for this application, we need to define an attribute map $\chi_4 : \mathbb{R}^{n^2-n} \times \mathbb{N} \to \mathbb{R}^s$, where the first argument is the type profile of all agents but agent 1, the second argument is the item assigned to agent 1, and s is a dimension of our choosing. A natural choice for χ_4 is:

$$\chi_4(\theta_{-1}, j) = (\theta_2[-j], \theta_3[-j], \ldots, \theta_n[-j]) \in \mathbb{R}^{(n-1)^2},$$

where $\theta_i[-j]$ denotes the vector obtained from θ_i by removing the jth entry. The attribute map thus reflects the agents' values for all items except item j, capturing the fact that the item assigned to agent 1 cannot be assigned to any other agent.

9.7 Experimental Evaluation

We perform a series of experiments to test our theoretical framework. To run our experiments, we use the SVM^{struct} package [Joachims et al., 2009], which allows for the use of custom kernel functions, attribute maps, and separation oracles.

9.7.1 Setup

We begin by briefly discussing our experimental methodology, performance metrics, and optimizations used to speed up the experiments.

Methodology

For each of the settings we consider, we generate three data sets: a training set, a validation set, and a test set. The training set is used as input to Training Problem 2, which in turn yields classifiers h_w and corresponding payment rules p_w. For each choice of the parameter C of Training Problem 2, and the parameter γ if the RBF kernel is used, a classifier h_w is learned based on the training set and evaluated based on the validation set. The classifier with the highest accuracy on the validation set is then chosen and evaluated on the test set. During training, we take the perspective of agent 1, and so a training set size of ℓ means that we train an SVM on ℓ examples. Once a partial outcome rule has been learned, however, it can be used to infer payments for all agents. We exploit this fact during testing, and report performance metrics across all agents for a given instance in the test set.

Metrics

We employ three metrics to measure the performance of the learned classifiers. These metrics are computed over the test set $\{(\theta^k, o^k)\}_{k=1}^{\ell}$.

Classification Accuracy Classification accuracy measures the accuracy of the trained classifier in predicting the outcome. Each instance of the ℓ instances has n agents, so in total we measure accuracy over $n\ell$ instances:[14]

$$accuracy = 100 \cdot \frac{\sum_{k=1}^{\ell} \sum_{i=1}^{n} \mathbb{I}(h_w(\theta_i, \theta_{-i}) = o_i^k))}{n\ell}.$$

Ex Post Regret We measure ex post regret by summing over the ex post regret experienced by all agents in each of the ℓ instances in the dataset, i.e.,

$$regret = \frac{\sum_{k=1}^{\ell} \sum_{i=1}^{n} rgt_i(\theta_i^k, \theta_{-i}^k)}{n\ell}.$$

[14]For a given instance θ, there are actually many ways to choose (θ_i, θ_{-i}) depending on the ordering of all agents but agent i. We discuss a technique we refer to as sorting in Section 9.7.1, which will choose a particular ordering. When this technique is not used, for example in application to the assignment problem, we fix an ordering of the other agents for each agent i, and use the same ordering across all instances.

Individual rationality violation This metric measures the fraction of individual rationality violation across all agents:

$$\textit{ir-violation} = \frac{\sum_{k=1}^{\ell}\sum_{i=1}^{n}\mathbb{I}(irv_i(\theta_i,\theta_{-i})>0)}{n\ell}.$$

Optimizations

In the case of multi-minded CAs, we first map the inputs θ_{-1} into a smaller space, which allows us to learn more effectively with smaller amounts of data. The barrier to using more data is not the availability of the data itself, but the time required for training, because training time scales quadratically in the size of the training set due to the use of non-linear kernels. For this step, we use *instance-based normalization*, which normalizes the values in θ_{-1} by the highest observed value and then rescales the computed payment appropriately, and *sorting*, which orders agents based on bid values.

Before passing examples θ to the learning algorithm or learned classifier, they are normalized by a positive multiplier so that the value of the highest bid by agents other than agent 1 is exactly 1, before passing it to the learning algorithm or classifier. The values and the solution are then transformed back to the original scale before computing the payment rule p_w. This technique of *instance-based normalization* leverages the observation that agent 1's allocation depends on the relative values of the other agent's reports, so that scaling all reports by a factor does not affect the outcome chosen. We apply this to multi-minded CAs and the assignment problem, but not to our experiments on CAs with positive k-wise dependent valuations.

In the sorting step, instead of choosing an arbitrary ordering of agents in θ_{-i}, we choose a specific ordering based on the maximum value the agent reports. For example, in a single-item setting, this amounts to ordering agents by their value. In the multi-minded CA setting, agents are ordered by the value they report for their most desired bundle. The intuition behind sorting is that we can again decrease the space of possible θ_{-i} reports the learner sees and learn more quickly. In the single-item case, we know that the second-price payment rule only depends on the maximum value across all other agents, and sorting places this value in the first coordinate of θ_{-i}. We apply sorting to the assignment problem by ordering agents by their maximum value for any item. We do not apply sorting to our experiments with k-wise dependent valuations in CAs.

9.7.2 Single-Item Auction

As a sanity check, we first perform experiments in application to a single-item auction with the efficient outcome rule, where the agent with the highest bid receives the item. For the distribution D on value profiles, we simply draw each agent's value independently from a uniform distribution on $[0, 1]$. The outcome rule g allocates the item to the agent with the highest value. We use a training set size of 300, and validation and test set sizes of 1000. We use an RBF kernel and parameters $C \in \{10^4, 10^5\}$ and $\gamma \in \{0.01, 0.1, 1\}$.

In this case, we know that the associated payment function that makes (g, p) strategyproof is the second-price payment rule.

The results reported in Table 9.1 and Figure 9.3 are for the χ_1, χ_2 attribute maps, which can be applied to this setting by observing that single-item auctions are a special case of multi-minded CAs. In particular, letting $\underline{0}$ be the 0 vector of dimension $n-1$, $\chi_1(\theta_{-1}, o_1) = (\theta_{-1}, \underline{0})$ if $o_1 = \emptyset$ and $\chi_1(\theta_{-1}, o_1) = (\underline{0}, \theta_{-1})$ if $o_1 = \{1\}$ and $\chi_2(\theta_{-1}, o_1) = \theta_{-1}$ if $o_1 = \emptyset$ and $\chi_2(\theta_{-1}, o_1) = \underline{0}$ if $o_1 = \{1\}$. For both choices of the attribute map we obtain excellent accuracy and very close approximation to the second-price payment rule. This shows that the framework is able to automatically learn the payment rule of Vickrey's auction (and despite the non-linearity in learning to price the item at what is effectively a maximum over the values of other agents.)

Table 9.1: Performance metrics for single-item auction.

n	accuracy		regret		ir-violation	
	χ_1	χ_2	χ_1	χ_2	χ_1	χ_2
2	99.7	93.1	0.000	0.003	0.00	0.07
3	98.7	97.6	0.000	0.000	0.01	0.00
4	98.4	99.1	0.000	0.000	0.00	0.01
5	97.3	96.6	0.001	0.001	0.02	0.00
6	97.6	97.4	0.000	0.001	0.00	0.02

9.7.3 Multi-Minded CAs

Type Distribution

Recall that in a multi-minded setting, there are m items, and each agent is interested in exactly $\kappa > 1$ bundles. For each bundle, we use the following procedure to determine which items are included in the bundle. We first assign an item to the bundle uniformly at random. Then with probability α, we add another random item (chosen uniformly from the remaining items), and with probability $(1 - \alpha)$ we stop. We continue this procedure

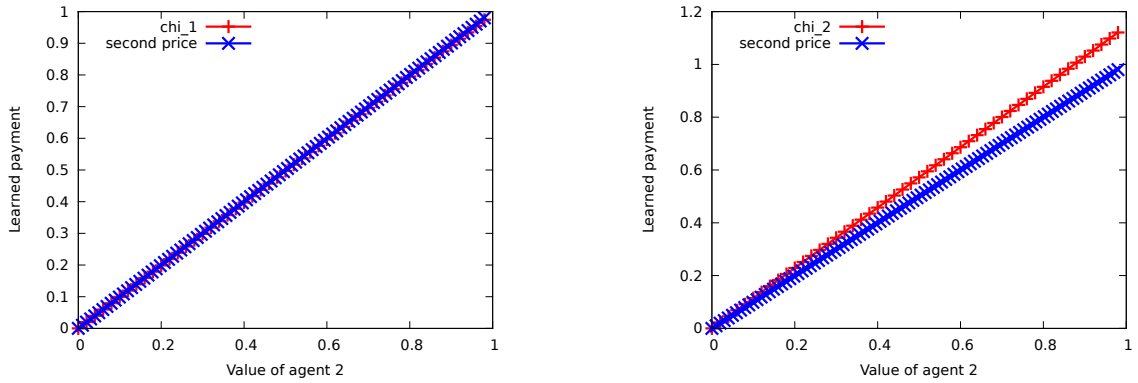


Figure 9.3: Learned payment rule vs. second-price payment rule for single-item auction with 2 agents, for χ_1 (left) and χ_2 (right).

until we stop or have exhausted the items. This procedure is inspired by Sandholm's decay distribution for the single-minded setting Sandholm [2002], and we use $\alpha = 0.75$ to be consistent with that setting, where this parameter value generated harder instances of the winner determination problem.

Once the bundle identities have been determined, we sample values for these bundles. Let c be an m-dimensional vector with entries chosen uniformly from $(0, 1]$. For each agent i, let d_i be an m-dimensional vector with entries chosen uniformly from $(0, 1]$. Each entry of c denotes the common value of a specific item, while each entry of d_i denotes the private value of a specific item for agent i. The value of bundle S_{ij} is then given by

$$v_{ij} = \min_{S_{ij'} \subseteq S_{ij}} \left(\frac{\langle S_{ij'}, \beta c + (1-\beta) d_i \rangle}{m} \right)^{\zeta}$$

for parameters $\beta \in [0, 1]$ and $\zeta > 1$. The inner product in the numerator corresponds to a sum over values of items, where common and private values for each item are respectively weighted with β and $(1 - \beta)$. The denominator normalizes all valuations to the interval $(0, 1]$. Parameter ζ controls the degree of complementarity among items: $\zeta > 1$ implies that goods are complements, whereas $\zeta < 1$ means that goods are substitutes. Choosing the minimum over bundles $S_{ij'}$ contained in S_{ij} finally ensures that the resulting valuations are monotonic.

Outcome Rules

We use two outcome rules in our experiments on multi-minded CAs. For the *optimal* outcome rule g_{opt}, the payment rule p_{vcg} makes the mechanism (g_{opt}, p_{vcg}) strategyproof.

Under this payment rule, agent i pays the externality it imposes on other agents. That is,

$$p_{vcg,1}(\theta) = \left(\max_{o \in \Omega} \sum_{i \neq 1} v_i(\theta_i, o_i) \right) - \sum_{i \neq 1} v_i(\theta_i, g_{opt,i}(\theta)).$$

The second outcome rule with which we experiment is a generalization of the greedy outcome rule for single-minded CA Lehmann et al. [2002]. Our generalization of the greedy rule is as follows. Let θ be the agent valuations and $o_i(j)$ denote the jth bundle desired by agent i. For each bundle $o_i(j)$, assign a score $v_i(\theta_i, o_i(j))/\sqrt{|o_i(j)|}$, where $|o_i(j)|$ indicates the total items in bundle $o_i(j)$. The greedy outcome rule orders the desired bundles by this score, and takes the bundle $o_i(j)$ with the next highest score as long as agent i has not already been allocated a bundle and $o_i(j)$ does not contain any items already allocated. While this greedy outcome rule has an associated payment rule that makes it strategyproof in the single-minded case, it is not implementable in the multi-minded case, as evidenced by the example provided in the appendix of this chapter.

Description of Experiments

We experiment with training sets of sizes 100, 300, and 500, and validation and test sets of size 1000. All experiments we report on are for a setting with 5 agents, 5 items, and 3 bundles per agent, and use $\beta = 0.5$ to generate the valuations, the RBF kernel, and parameters $C \in \{10^4, 10^5\}$ and $\gamma \in \{0.01, 0.1, 1\}$.

Basic Results

Table 9.2 presents the basic results for multi-minded CAs with optimal and greedy outcome rules, respectively. For both outcome rules, we present the results for p_{vcg} as a baseline. Because p_{vcg} is the strategyproof payment rule for the optimal outcome rule, p_{vcg} always has accuracy 100, regret 0, and IR violation 0 for the optimal outcome rule. The main findings are that our learned payment rule has low regret for the optimal outcome rule and regret that is about the same as or better than the regret of p_{vcg} when the outcome rule is greedy. Given that greedy winner determination is seeking to maximize total welfare it is natural the VCG-based payments would perform reasonably well in this environment.

Across all instances, as expected, accuracy is negatively correlated with regret and ex post IR violation. The degree of complementarity between items, ζ, as well as the outcome rule chosen, has a major effect on the results. Instances with low complementarity ($\zeta = 0.5$) yield payment rules with higher regret, and χ_1 performs better on the greedy outcome

Table 9.2: Results for multi-minded CA with training set size 500.

		Optimal outcome rule									Greedy outcome rule								
		accuracy			regret			ir-violation			accuracy			regret			ir-violation		
n	ζ	p_{vcg}	χ_1	χ_2	p_{vcg}	χ_1	χ_2	p_{vcg}	χ_1	χ_2	p_{vcg}	χ_1	χ_2	p_{vcg}	χ_1	χ_2	p_{vcg}	χ_1	χ_2
2	0.5	100	70.7	91.9	0	0.014	0.002	0.0	0.06	0.03	50.9	59.1	40.6	0.079	0.030	0.172	0.22	0.12	0.33
3	0.5	100	54.5	75.4	0	0.037	0.017	0.0	0.19	0.10	55.4	57.9	54.7	0.070	0.030	0.088	0.18	0.21	0.36
4	0.5	100	53.8	67.7	0	0.042	0.031	0.0	0.22	0.18	61.1	58.2	57.9	0.056	0.033	0.056	0.14	0.20	0.31
5	0.5	100	15.8	67.0	0	0.133	0.032	0.0	0.26	0.19	64.9	61.3	63.0	0.048	0.027	0.042	0.13	0.19	0.24
6	0.5	100	61.1	68.2	0	0.037	0.032	0.0	0.22	0.20	66.6	63.8	63.8	0.041	0.034	0.045	0.12	0.20	0.24
2	1.0	100	84.5	93.4	0	0.008	0.001	0.0	0.08	0.02	87.8	86.6	84.0	0.007	0.005	0.008	0.04	0.06	0.09
3	1.0	100	77.1	83.5	0	0.012	0.005	0.0	0.13	0.09	85.3	86.7	85.7	0.006	0.006	0.006	0.04	0.07	0.05
4	1.0	100	74.6	81.1	0	0.014	0.009	0.0	0.16	0.12	82.4	86.5	84.2	0.006	0.006	0.007	0.05	0.08	0.08
5	1.0	100	73.4	77.4	0	0.018	0.011	0.0	0.19	0.12	82.7	85.8	84.9	0.007	0.009	0.009	0.04	0.10	0.10
6	1.0	100	75.0	77.7	0	0.020	0.013	0.0	0.20	0.16	80.0	87.4	88.1	0.006	0.007	0.005	0.04	0.08	0.07
2	1.5	100	91.5	96.9	0	0.004	0.000	0.0	0.06	0.02	94.7	91.1	91.7	0.002	0.002	0.002	0.02	0.04	0.04
3	1.5	100	91.0	93.4	0	0.004	0.001	0.0	0.05	0.03	97.1	92.8	93.2	0.001	0.002	0.001	0.01	0.02	0.04
4	1.5	100	92.5	94.2	0	0.003	0.001	0.0	0.03	0.04	96.4	91.5	92.1	0.001	0.003	0.002	0.02	0.07	0.07
5	1.5	100	91.7	93.9	0	0.004	0.002	0.0	0.06	0.03	97.5	90.5	91.4	0.001	0.004	0.002	0.01	0.06	0.04
6	1.5	100	91.9	93.7	0	0.003	0.001	0.0	0.05	0.04	98.4	92.2	92.8	0.000	0.003	0.002	0.01	0.06	0.06

rule while χ_2 performs better on the optimal outcome rule. For high complementarity between items the greedy outcome tends to allocate all items to a single agent, and the learned price function sets high prices for small bundles to capture this property. For low complementarity the allocation tends to be split and less predictable. Still, the best classifiers achieve average ex post regret of less than 0.032 (for values normalized to [0,1]) even though the corresponding prediction accuracy can be as low as 67%.

For the greedy outcome rule, the performance of p_{vcg} is comparable for $\zeta \in \{1.0, 1.5\}$ but worse than the payment rule learned in our framework in the case of $\zeta = 0.5$, where the greedy outcome rule becomes less optimal.

Effect of Training Set Size

Table 9.3 charts performance as the training set size is varied for the greedy outcome rule. While training data is readily available (we can simply sample from D and run the outcome rule g), training time becomes prohibitive for larger training set sizes. Table 9.3 shows that regret decreases with larger training sets, and for a training set size of 500, the best of χ_1 and χ_2 outperforms p_{vcg} for $\zeta = 0.5$ and is comparable to p_{vcg} for $\zeta \in \{1.0, 1.5\}$.

Table 9.3: Effect of training set size on accuracy of learned classifier. Multi-minded CA, greedy outcome rule. Training set size is given in the column labels for χ_1, χ_2. p_{vcg} does not have a training set size.

n	ζ	accuracy	100		300		500		regret	100		300		500	
		p_{vcg}	χ_1	χ_2	χ_1	χ_2	χ_1	χ_2	p_{vcg}	χ_1	χ_2	χ_1	χ_2	χ_1	χ_2
2	0.5	50.9	54.3	48.2	57.0	46.9	59.1	40.6	0.079	0.045	0.195	0.032	0.098	0.030	0.172
3	0.5	55.4	50.1	49.8	55.7	54.4	57.9	54.7	0.070	0.054	0.078	0.038	0.082	0.030	0.088
4	0.5	61.1	53.4	56.2	56.4	58.5	58.2	57.9	0.056	0.050	0.059	0.040	0.061	0.033	0.056
5	0.5	64.9	14.2	57.9	61.0	61.8	61.3	63.0	0.048	0.173	0.064	0.038	0.048	0.027	0.042
6	0.5	66.6	58.4	58.8	62.2	63.9	63.8	63.8	0.041	0.039	0.059	0.037	0.049	0.034	0.045
2	1.0	87.8	80.7	80.5	84.4	84.1	86.6	84.0	0.007	0.010	0.010	0.009	0.008	0.005	0.008
3	1.0	85.3	74.9	78.0	83.0	80.6	86.7	85.7	0.006	0.020	0.011	0.009	0.009	0.006	0.006
4	1.0	82.4	78.5	80.1	84.2	83.1	86.5	84.2	0.006	0.015	0.014	0.008	0.009	0.006	0.007
5	1.0	82.7	81.0	81.8	84.3	84.3	85.8	84.9	0.007	0.020	0.014	0.010	0.009	0.009	0.009
6	1.0	80.0	81.8	83.7	87.6	88.3	87.4	88.1	0.006	0.062	0.018	0.008	0.005	0.007	0.005
2	1.5	94.7	83.3	88.1	89.3	89.8	91.1	91.7	0.002	0.008	0.003	0.003	0.002	0.002	0.002
3	1.5	97.1	86.9	87.6	90.3	91.5	92.8	93.2	0.001	0.005	0.004	0.003	0.002	0.002	0.001
4	1.5	96.4	88.4	90.7	89.3	90.8	91.5	92.1	0.001	0.005	0.003	0.004	0.003	0.003	0.002
5	1.5	97.5	87.2	88.5	91.4	90.5	90.5	91.4	0.001	0.006	0.004	0.003	0.003	0.004	0.002
6	1.5	98.4	86.3	86.8	91.4	92.5	92.2	92.8	0.000	0.011	0.007	0.004	0.002	0.003	0.002

Table 9.4: Impact of payment offset and null loss fix for $\zeta = 0.5$ and greedy outcome rule, training set size 300. All results are for χ_2, null loss values appear in the second row.

payment	accuracy			regret			ir-violation			ir-fix-welfare-avg		
offset	0.5	1.0	1.5	0.5	1.0	1.5	0.5	1.0	1.5	0.5	1.0	1.5
0	59.7	61.8	61.7	0.065	0.048	0.042	0.35	0.26	0.21	0.27	0.43	0.52
0.05	61.7	61.2	60.1	0.054	0.045	0.044	0.29	0.20	0.15	0.37	0.54	0.65
0.10	62.1	59.3	56.7	0.048	0.047	0.051	0.23	0.14	0.10	0.48	0.66	0.75
0.15	60.4	55.1	52.2	0.047	0.055	0.064	0.17	0.10	0.06	0.59	0.75	0.84
0.20	57.8	51.7	48.5	0.052	0.067	0.079	0.12	0.06	0.03	0.70	0.83	0.90
0.25	54.3	47.7	44.3	0.061	0.082	0.096	0.08	0.03	0.02	0.79	0.89	0.93

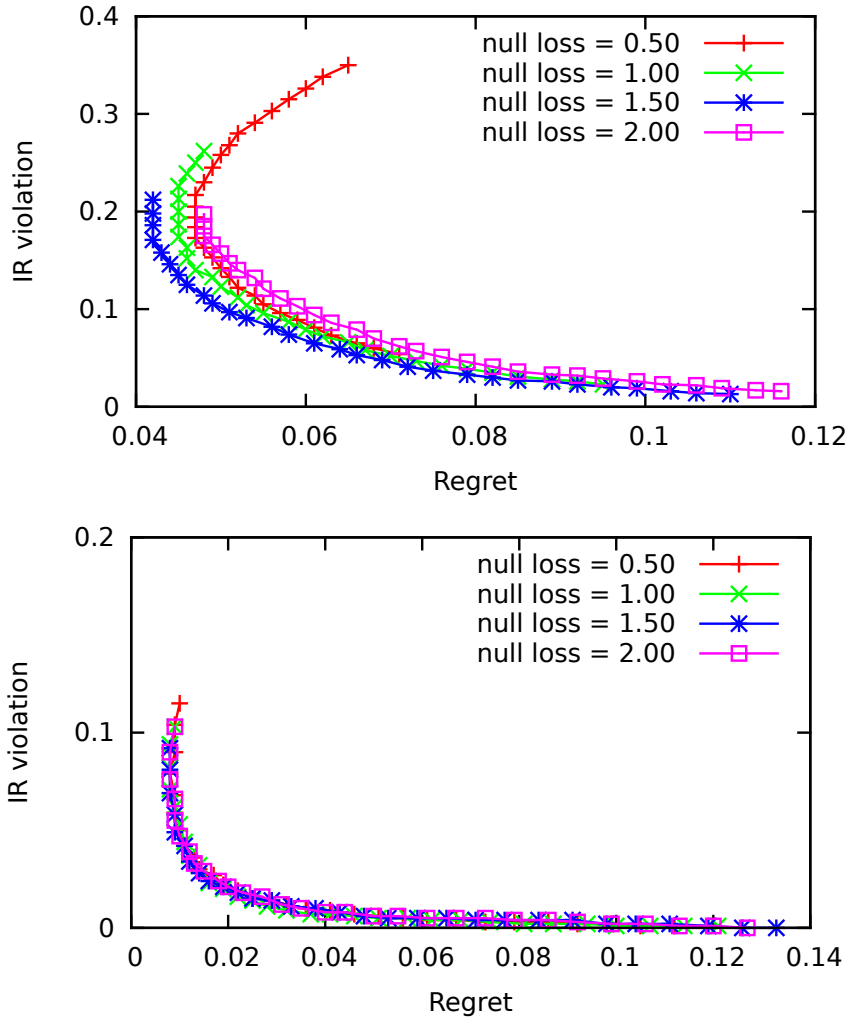


Figure 9.4: Impact of payment offset and null loss fix for greedy outcome rule, training set size 300.

IR Fixes

Table 9.4 summarizes our results regarding the various fixes to IR violations, for the particularly challenging case of the greedy outcome rule and $\zeta = 0.5$. The extent of IR violation decreases with larger payment offset and null loss. Regret tends to move in the opposite direction, but there are cases where IR violation and regret both decrease. The three rightmost columns of Table 9.4 list the average ratio between welfare after and before the deallocation fix, across the instances in the test set. With a payment offset of 0, a large welfare hit is incurred if we deallocate agents with IR violations. However, this penalty decreases with increasing payment offsets and increasing null loss. At the most extreme payment offset and null loss adjustment, the IR violation is as low as 2%, and the deallocation fix incurs a welfare loss of only 7%.

Figure 9.4 shows a graphical representation of the impact of payment offsets and null losses. Each line in the plot corresponds to a payment rule learned with a different null loss, and each point on a line corresponds to a different payment offset. The payment offset is zero for the top-most point on each line, and equal to 0.29 for the lowest point on each line. Increasing the payment offset always decreases the rate of IR violation, but may decrease or increase regret. Increasing null loss lowers the top-most point on a given line, but arbitrarily increasing null loss can be harmful. Indeed, in the figure on the left, a null loss of 1.5 results in a slightly higher top-most point but significantly lower regret at this top-most point compared to a null loss of 2.0. It is also interesting to note that these adjustments have much more impact on the hardest distribution with $\zeta = 0.5$.

Item Monotonicity

Table 9.5 presents a comparison of a payment rule learned with explicit enumeration of all bundle constraints (the default that we have been using for our other results) and a payment rule learned by optimistically assuming item monotonicity (see Section 9.6.1). Performance is affected when we drop constraints and optimistically assume item monotonicity, although the effects are small for $\zeta \in \{1.0, 1.5\}$ and larger for $\zeta = 0.5$. Because item monotonicity allows for the training problem to be succinctly specified, we may be able to train on more data, and this seems a very promising avenue for further consideration (perhaps coupled with heuristic methods to add additional constraints to the training problem).

Table 9.5: Comparison of performance with and without optimistically assuming item monotonicity. (i-mon) indicates a payment rule learned by optimistically assuming item monotonicity. greedy outcome rule. Training set size 300.

n	ζ	accuracy		regret		ir-violation	
		χ_2	χ_2 (i-mon)	χ_2	χ_2 (i-mon)	χ_2	χ_2 (i-mon)
2	0.5	46.9	46.3	0.098	0.232	0.28	0.38
3	0.5	54.4	8.6	0.082	0.465	0.33	0.06
4	0.5	58.5	48.2	0.061	0.811	0.31	0.25
5	0.5	61.8	57.0	0.048	0.136	0.26	0.26
6	0.5	63.9	61.3	0.049	0.078	0.25	0.20
2	1.0	84.1	82.2	0.008	0.010	0.06	0.08
3	1.0	80.6	80.1	0.009	0.010	0.10	0.09
4	1.0	83.1	79.7	0.009	0.012	0.11	0.11
5	1.0	84.3	77.2	0.009	0.020	0.10	0.11
6	1.0	88.3	83.9	0.005	0.013	0.08	0.11
2	1.5	89.8	89.1	0.002	0.003	0.03	0.06
3	1.5	91.5	91.3	0.002	0.003	0.04	0.04
4	1.5	90.8	89.7	0.003	0.003	0.06	0.06
5	1.5	90.5	87.3	0.003	0.005	0.04	0.05
6	1.5	92.5	70.8	0.002	0.081	0.06	0.17

9.7.4 Combinatorial Auctions with Positive k-wise Dependent Valuations

We experiment with our framework on combinatorial auctions with positive k-wise dependent valuations. We find that our learned payment rules can outperform VCG-based payment rules in terms of regret for settings with large numbers of items, and outperform VCG-based payment rules in terms of the trade-off between IR violation and regret. Because we have a formulation of the separation problem as a small LP as discussed in Section 9.6.2, we are able to train payment rules and compute regret for larger instances.

In order to experiment with positive k-wise dependent valuations in combinatorial auctions, we need a way to generate such valuations. To construct an agent's valuation, we first specify the nodes and edges in a graph (V, E), and then assign weights $g(v)$ and $g(e)$ over the nodes and edges. For every possible edge (j_1, j_2), we add the edge to the agents' graph with probability ρ. $g(v)$ is sampled uniformly at random from $[0, 1]$; the weight for each added edge is also sampled uniformly at random from $[0, 1]$. With this setup, the edge probability parameter ρ lets us generate test instances of varying edge density. So that our regret numbers are comparable across different size instances, we normalize each agent's weights by the expected value for the set of all items.

The outcome rule we use is GREEDY-KWISE outlined in Section 9.6.2. We use a training set size of 1000 and validation and test sets of size 500. While in the experiments for multi-

Table 9.6: Basic results for valuations with $\rho = 0.1$. Metrics for 10 and 20 items are computed using an approximation based on the tractable separation oracle for our training problem. Metrics are not computed for VCG-based rules because computation requires brute force enumeration over all possible bundles(but can be efficiently computed for the succinctly represented, trained payment rule).

agents	items	accuracy		regret		ir-violation	
		χ	p_{vcg}	χ	p_{vcg}	χ	p_{vcg}
6	2	94.9	97.3	0.006	0.008	0.025	0.008
6	4	70.7	82.7	0.020	0.022	0.146	0.045
6	6	53.1	66.4	0.027	0.031	0.246	0.082
6	10	28.3	–	0.033	–	0.442	–
6	20	–	–	–	–	–	–

minded CA and egalitarian assignment a sample type profile is converted into a single training point, in these experiments we convert a single sample type profile into n training points, one corresponding to each of the n agents serving as agent 1. In running our experiments, we noticed that training in this way improves testing performance (in the testing phase, each sample type profile is converted into n different testing points). For this setting, it appears that training on just a single agent for each sampled type profile over-emphasizes a single agent and learns a payment rule with worse performance when applied to all agents. We did not observe this in the other two settings (and it would have been difficult to do this since the training problem does not scale well), but gaining a better understanding of this phenomenon is a direction for future work.

We compare against a VCG-based payment rule which runs GREEDY-KWISE on all agents and GREEDY-KWISE on all agents excluding agent i and charges agent i the difference in value to agents other that i in the two allocations.

Tables 9.6 and 9.7 and Figure 9.7.4 compare our learned payment rules (with 0 null loss) to the VCG-based payment rule for $\rho = 0.1$ and $\rho = 0.9$. The learned payment rule has better regret, despite having worse accuracy. However, the learned payment rule incurs significant IR violation.

We examine the IR violation issue in Figure 9.7.4. Here we implement the two IR fixes of increasing the null loss value and applying payment offsets. We see that across all instances, we can find settings of the null loss for which our IR / regret curve lies beneath that of the VCG-based payment rule, indicating that we have settings which have better regret and lower IR violation. We also see that despite having significant IR failures when we have no payment offset, we can significantly decrease IR violation at the cost of a small amount of regret increase by using a payment offset.

Table 9.7: Basic results for valuations with $\rho = 0.9$. Metrics for 10 and 20 items are computed using an approximation based on the tractable separation oracle for our training problem. Metrics are not computed for VCG-based rules because computation requires brute force enumeration over all possible bundles (but can be efficiently computed for the succinctly represented, trained payment rule).

agents	items	accuracy		regret		ir-violation	
		χ	p_{vcg}	χ	p_{vcg}	χ	p_{vcg}
6	2	79.9	82.1	0.024	0.028	0.108	0.048
6	4	64.0	51.8	0.038	0.056	0.227	0.118
6	6	61.6	42.7	0.030	0.062	0.229	0.129
6	10	61.6	–	0.026	–	0.238	–
6	20	–	–	–	–	–	–

9.7.5 The Egalitarian Assignment Problem

In the assignment problem, agents' values for the items are sampled uniformly and independently from $[0, 1]$. We use a training set of size 600, validation and test sets of size 1000, and the RBF kernel with parameters $C \in \{10, 1000, 100000\}$ and $\gamma \in \{0.1, 0.5, 1.0\}$. We find that our learned payment rules have significantly better accuracy and regret than VCG-based payment rules. We explain the improvement over VCG-based payments by observing that the egalitarian rule is not maximizing total welfare, and thus not compatible in this sense with VCG-based ideas.

The performance of the learned payment rules is compared to that of three VCG-based payment rules. For this, let W be the total welfare of all agents other than i under the outcome chosen by g, and W_{eg} be the minimum value any agent other than i receives under this outcome. We consider the following payment rules:

(1) the *vcg* payment rule, where agent i pays the difference between the maximum total welfare of the other agents under any allocation and W;

(2) the *tot-vcg* payment rule, where agent i pays the difference between the total welfare of the other agents under the allocation maximizing egalitarian welfare and W; and

(3) the *eg-vcg* payment rule, where agent i pays the difference between the minimum value of any agent under the allocation maximizing egalitarian welfare and W_{eg}.

The results for attribute map χ_4 are shown in Table 9.8. We see that the learned payment rule p_w yields significantly lower regret than any of the VCG-based payment rules, and average ex post regret less than 0.074 for values normalized to $[0, 1]$. Since we are not maximizing the sum of values of the agents, it is not very surprising that VCG-based payment rules perform rather poorly. The learned payment rule p_w can adjust to the

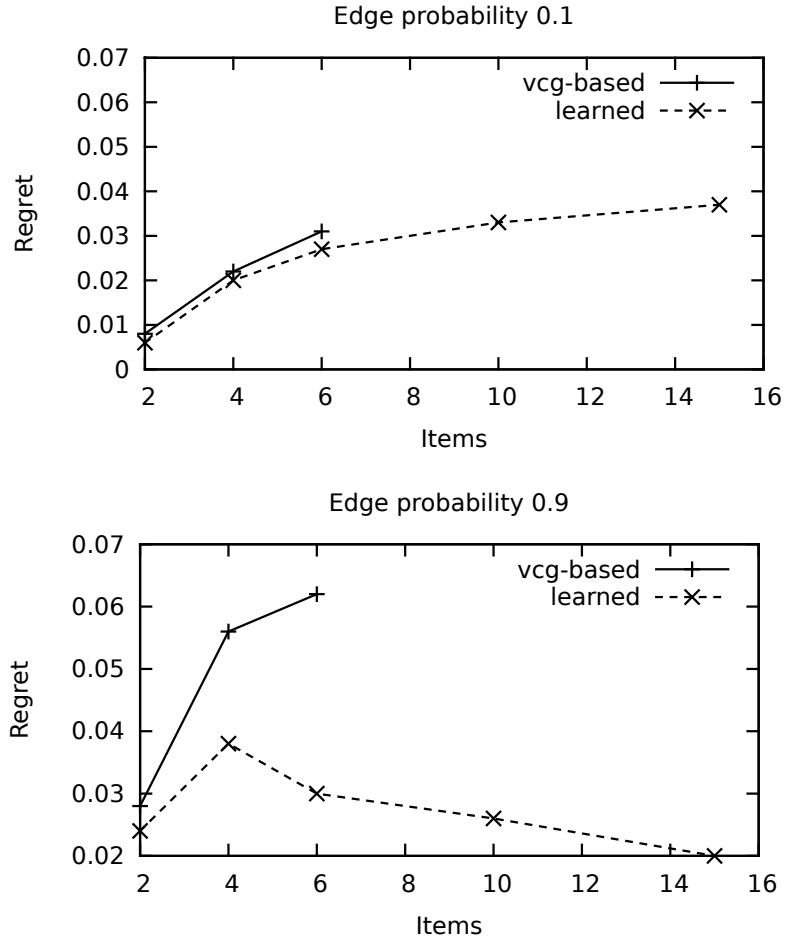


Figure 9.5: Regret v. Number of Items for learned payment rule and VCG-based payment rules. For 10 and 15 items, we do not have regret number for the VCG-based rules because computing regret requires enumeration over all possible bundles. In this case, the regret for learned payment rules and 10 and 15 items is an upper bound on the true regret obtained by applying our tractable separation oracle.

outcome rule, and also achieves a low fraction of ex post IR violation of at most 3%.

9.8 Summary and Future Work

Whereas in Chapter 8 we give a computational procedure that makes BnB search monotone and therefore truthful for known single-minded CAs, in this chapter we relax away from exact incentive compatibility and propose a framework that applies to general mechanism design settings. We introduce a new paradigm for computational mechanism design, in which statistical machine learning is adopted to design payment rules for outcome rules, and show encouraging experimental results. The mechanism design domain can be multi-

Table 9.8: Results for assignment problem with egalitarian outcome rule

n	accuracy				regret				ir-violation			
	vcg	tot-vcg	eg-vcg	p_w	vcg	tot-vcg	eg-vcg	p_w	vcg	tot-vcg	eg-vcg	p_w
2	64.3	67.5	67.5	89.0	0.018	0.015	0.015	0.023	0.03	0.01	0.01	0.03
3	48.0	52.1	42.5	77.9	0.070	0.077	0.127	0.041	0.06	0.07	0.03	0.04
4	40.6	43.1	30.8	71.0	0.111	0.123	0.199	0.054	0.07	0.09	0.03	0.02
5	32.4	35.3	24.5	63.9	0.157	0.169	0.254	0.071	0.10	0.12	0.03	0.01
6	27.1	29.9	20.0	59.0	0.189	0.208	0.290	0.074	0.10	0.13	0.03	0.01

parameter, and the outcome rules can be specified algorithmically and need not be designed for objectives that are separable across agents. Central to our approach is to relax incentive compatibility as a hard constraint on mechanism design, adopting in its place the goal of minimizing expected regret while requiring agent-independent prices.

Future Work

Future directions of interest include:

1. Considering alternative learning paradigms, including formulations of the problem as a regression rather than classification problem.

2. Gaining a better understanding of our results for single-minded CA in relation to Lahaie [2011]. Lahaie [2011] shows that only quadratic prices are needed to find market clearing prices for single-minded CAs. In light of these results, it would be interesting to do a more thorough examination of the trade-off between polynomial and RBF kernels when learning payment rules for single-minded CA. Though the market clearing problem differs from the problem of learning admissible, agent-independent payment rules, it does suggest that polynomial kernels are quite powerful and it may not be necessary to go all the way to RBF kernels.

3. Developing formulations that can impose constraints on properties of the learned payment rule, concerning for example the core, budgets, or individual-rationality properties.

4. Developing methods that learn classifiers more likely to induce feasible outcome rules, so that these learned outcome rules can be used directly.

5. Extending the approach to domains without money by developing a structure on discriminant functions appropriate to the incentive considerations facing rational self-interested agents in such domains.

6. Investigating the extent to which alternative goals such as regret percentiles or interim regret can be achieved through machine learning.

7. Continuing to explore succinct valuation representations in our method, perhaps by supporting the use of concise approximate valuations with additional kernel operators.

Appendix: Greedy Allocation Rule is not Weakly Monotone

Consider a setting with a single agent and four items.

If the valuations θ_1 of the agent are

$$v_1(\theta_1, o_1) = \begin{cases} 20 & \text{if } o_1 = \{1,2,3,4\} \\ 12 & \text{if } 1 \in o_1 \text{ and } j \notin o_1 \text{ for some } j \in \{2,3,4\}, \text{ and} \\ 0 & \text{else} \end{cases}$$

then the allocation is $\{1\}$.

If the valuations are θ_1' such that

$$v_1(\theta_1', o_1) = \begin{cases} 12 & \text{if } o_1 = \{1,2,3,4\} \\ 5 & \text{if } 1 \in o_1 \text{ and } j \notin o_1 \text{ for some } j \in \{2,3,4\}, \text{ and} \\ 0 & \text{else} \end{cases}$$

then the allocation is $\{1,2,3,4\}$.

We have $v_1(\theta_1', \{1,2,3,4\}) - v_1(\theta_1', \{1\}) < v_1(\theta_1, \{1,2,3,4\}) - v_1(\theta_1, \{1\})$ contradicting weak monotonicity.

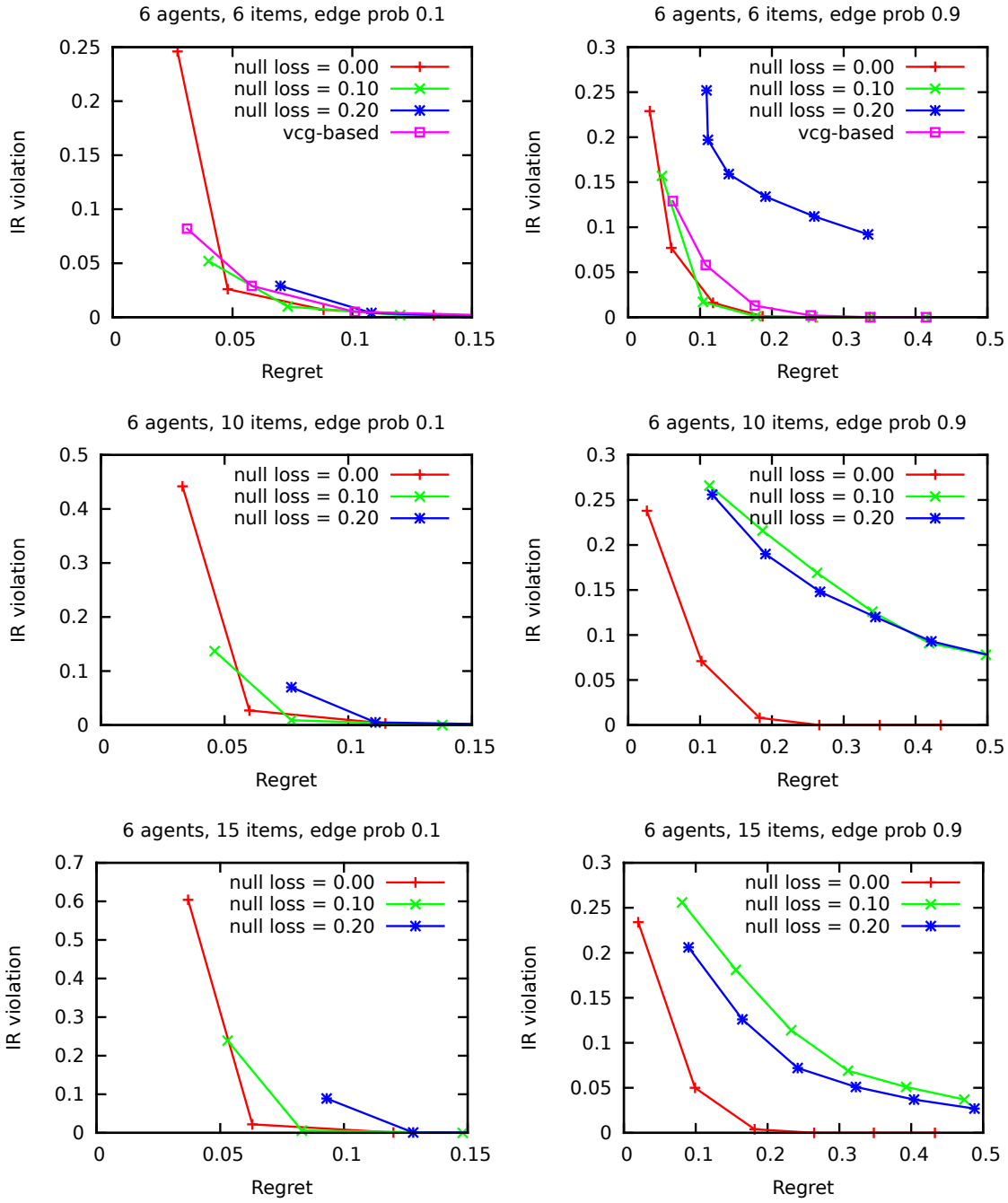


Figure 9.6: Regret v. IR Violation trade-off for learned payment rule and VCG-based payment rule for k-wise dependent valuations. We do not have regret numbers for the VCG-based rule and 10 and 20 items because computing regret requires brute force enumeration over all possible bundles. In this case, the regret numbers for the learned payment rule are an upper bound on regret obtained by using our tractable separation oracle.

Chapter 10

Conclusions

Resource allocation problems are abundant in modern day life. In this dissertation, we undertake the study of resource allocation procedures that satisfied the properties of fairness, truthfulness, or both. For fairness, we focus on the cake cutting problem of dividing a heterogeneous, divisible good. For truthfulness, we study both the cake cutting problem as well as combinatorial auctions (CAs). The approaches are inspired by thinking about computational aspects of resource allocation problems. For cake cutting, we consider issues of communication complexity and depart from the classic cake cutting setting where agent valuations cannot be succinctly communicated. Instead, we consider a direct revelation model under restricted valuations where valuations can be succinctly communicated to a decision maker. This minor change in perspective opens a large number of interesting questions, some of which we tackle in this dissertation. On the mechanism design side, we take computational approaches to mechanism design whereby our mechanisms cannot be specified by hand but rather are the result of some computational procedure. This helps us leverage effective *heuristic* methods for the purposes of mechanism and circumvent the analytical difficulties associated with complex mechanism design settings.

The computer science approaches outlined above are important to bridging the gap between theory and practice. The assumptions on cake cutting, while restrictive, allow for stronger results and also have the potential to create algorithms that have more natural interactions with agents. The contributions to computational approaches to mechanism design give us approaches for dealing with complex settings while keeping in mind the importance of computational tractability. We believe the lens of computation can facilitate the adoption and use of research on resource allocation in practice.

10.1 Brief Review

10.1.1 Cake Cutting with Restricted Valuations

The first part of this dissertation examines the cake cutting problem of fairly dividing a heterogeneous, divisible good. In Chapter 3, we introduce the cake cutting problem and discuss classic results from the literature. We then describe the focus of the cake cutting work in this dissertation, introducing a direct revelation model and restricted families of valuations. The most restrictive family of valuations, *piecewise uniform valuations*, capture the situation where intervals of cake are either desired or undesired, with the marginal value being the same across all desired intervals. We also introduce the more general classes of *piecewise constant* and *piecewise linear* valuations. Though these are restrictive, they are significantly more general than piecewise uniform valuations and can be used to attain close approximations of general valuations. Though they are less expressive, the key advantage to these valuations is that they can be succinctly communicated. In the most general case of piecewise linear valuations, an agent can specify the intervals on which its density function is linear and provide the slope and intercept of the density function on each of these intervals. Succinct representation allows for a new paradigm akin to the direct revelation model in mechanism where cake cutting algorithms operate directly on agents' exact valuations. The contributions in this section of the dissertation assume these restricted families of valuations and a direct revelation model. Specifically, the main contributions are:

- **Algorithms for welfare maximization.** In Section 4.2 we design algorithms that find *maxsum* fair allocations, i.e. allocations that have the best social welfare among a set of fair allocations. We provide a linear programming approach for the case of piecewise constant valuations. For piecewise linear valuations, we prove that exactly computing a maxsum EF allocation is impossible since there are cases where all maxsum EF allocations require cuts at irrational endpoints, even if the agents' valuations are specified by rational numbers. Circumventing this, we provide an algorithm for two agents and piecewise linear valuations that finds an allocation with envy at most ϵ and welfare at least as great as any maxsum EF allocation. This algorithm runs in time polynomial in $\log(1/\epsilon)$. We also provide an algorithm for more general valuations that finds an allocation with envy at most ϵ and welfare at most ϵ less than any maxsum EF allocation in time polynomial in $1/\epsilon$.

- **Analysis of maxsum fair allocations.** In Section 4.3 we analyze the properties of maxsum fair allocations. The main findings are that maxsum EF allocations may

not be Pareto-efficient and that the welfare of maxsum EF allocations are at least as great as the welfare of maxsum EQ allocations when agents have piecewise linear valuations. These findings shed light on the question on the quality of maxsum fair allocations and how maxsum fair allocations differ for various notions of fairness.

- **More expressive cake cutting.** One potential barrier to the application of cake cutting algorithms is the assumption that agent valuations are completely additive. That is, agents receive positive value even if they are given a set of very small, disjoint intervals. In Chapter 5, we study the special case of piecewise uniform valuations, but allow agents to have a minimum length parameter λ that specifies that intervals less than a certain length yield no value. Exact proportionality is not possible in this setting, and we give approximately proportional algorithms that are essentially optimal for this setting. We also investigate approximate proportionality and envy-freeness together and give an intricate algorithm that finds an approximately proportional and EF allocation for two agents.

- **Truthful cake cutting.** The previous results assume that agents truthfully report their valuations, or alternatively, that valuations are publicly known to the cake cutting algorithm. In Chapter 6, we investigate cake cutting under the assumption that agents are strategic and may misreport their preferences if it is beneficial. The main result is a DSIC, proportional, EF, Pareto-efficient, and polynomial time deterministic mechanism for any number of agents with piecewise uniform valuations. The result depends on a particular network flow graph and application of the max-flow min-cut theorem for polynomial time computation and to prove incentive compatibility. The mechanism resembles the probabilistic serial and simultaneous eating mechanisms for the random assignment problem. We also give randomized mechanisms. These mechanisms handle piecewise linear valuations, but are only truthful in expectation (over the randomness of the mechanism).

10.1.2 Computational Approaches to Mechanism Design

The second part of this dissertation takes two different computational approaches to mechanism design. While classical mechanism design in economics typically seeks an analytic description of mechanisms, we adopt approaches that produce mechanisms as the result of a computational process. This allows us to leverage sophisticated, heuristic algorithms for the purposes of mechanism design and also design mechanisms for complex, multi-parameter

settings that are difficult to analyze or reason about theoretically. We have two specific contributions:

- **Monotone branch and bound (BnB) search.** In Chapter 8, we take the approach of heuristic mechanism design [Parkes, 2009], using BnB search as the heuristic algorithm in application to known single-minded CAs. We design a procedure that performs sensitivity analysis on the BnB search tree, deallocating items when we find failures of monotonicity. This yields a modified BnB search procedure that is monotone in the agents' reported values for their publicly known target bundle. We implement and test monotone BnB search on distributions from the literature and find settings where monotone BnB search produces welfare better than existing monotone approximation algorithms while exhibiting better runtime (including the overhead introduced by sensitivity checking) than running BnB search to optimality.

- **Learning payment rules.** While our work on BnB search treats incentive compatibility as a hard constraint and therefore modifies the outcome rule to attain exact incentive compatibility, in Chapter 9 we examine a dual approach where we take the outcome rule as fixed and seek a payment rule that minimizes agent incentives to misreport. We draw a novel connection between incentive compatible mechanisms and a particular type of multi-class classifier. The specific classification problem we solve is to predict the outcome rule given an agent's reported valuations. A payment rule can be derived from the form of the learned classifier. An exact classifier gives an incentive compatible payment rule, while a classifier that minimizes a carefully defined error function produces a payment rule that minimizes *expected ex-post regret*, where the expectation is take over valuations drawn from a commonly known distribution. We implement the techniques using structural support vector machines and apply the framework to multi-minded CAs, positive k-wise dependent valuations, and the assignment problem where the outcome rule maximizes egalitarian welfare. In each of these settings we find that the learned payment rules have expected regret better than the VCG-inspired baselines we compare against.

10.2 Future Directions

In this section I outline some possible future directions related to the topics discussed in this dissertation. While each chapter contains specific areas for future work, I view the issues listed below as more high-level directions and agendas to be pursued.

Restricted valuations in the classic model of cake cutting. Our work on cake cutting focuses on a direct revelation model where agents have restricted families of valuations. It is interesting to think about whether we can obtain stronger results in the classic model of cake cutting while assuming restricted families of valuations. Recent work by Kurokawa et al. [2013] takes a step in this direction by showing that a bounded EF algorithm for piecewise uniform valuations in the classic cake cutting model would imply a bounded EF algorithm for general valuations.

Truthfulness, fairness, and efficiency. The main open question from Chapter 6 is whether we can obtain truthful, proportional, and EF mechanisms when agent valuations are piecewise constant (rather than piecewise uniform). As discussed in that chapter, previous results show that truthfulness and Pareto-efficiency are not possible [Schummer, 1997], but truthfulness and fairness may still be possible.

We can also consider truthful and fair (EF) CAs. Several papers consider whether truthful, EF, and welfare-maximizing mechanisms exist for different classes of valuations [Pápai, 2003, Cohen et al., 2011, Feldman and Lai, 2012]. In particular, Feldman and Lai [2012] show that such mechanisms do not exist for a subclass of sub-additive valuations. Given this impossibility, a natural question is to consider mechanisms that relax exact truthfulness, fairness, or welfare-maximization.

Approximate incentive compatibility. Can we extend the framework for learning payment rules to other notions of approximate incentive compatibility? The current framework learns a rule that minimizes expected ex-post regret, averaged across all agents given a random draw from the distribution of types. A different measure to minimize would be interim regret. Fixing an agent's type, interim regret compares an agent's expected utility under truthful reports to the agent's expected utility under the best possible report, where the expectation is taken over other agent's types conditional on drawing the agent's type. We could minimize expected interim regret where this regret is averaged over a draw of the agent's type from the distribution (this is a weaker notion than expected ex post regret), or we could try to minimize a threshold γ that guarantees that interim regret is at most γ. These alternate measures of approximate incentive compatibility do not map nicely to existing machine learning approaches, but it would be interesting to try and develop methods to learn payment rules that minimize these measures.

Along these same lines, it would be interesting to do some empirical work on appoximate incentive compatibility. Specifically, in Chapter 9 the payment rules we learn have an *agent independence* property. Is agent independence enough to ensure truthful reports in practice?

How do agents behave when participating in mechanisms with payment rules that satisfy agent independence but are not fully incentive compatible? Is there some way to quantify how incentive compatible such mechanisms are? For instance, if deviations are possible but require large changes in reports, intuitively these mechanisms will be less vulnerable to manipulation. Similarly, though Chapter 9 focuses on finding payment rules with small regret, it might be the case that even small amounts of regret cause agents to deviate, thereby changing the distribution that other agents face, thereby causing the mechanism to unravel. It would be nice to quantify the robustness of the learned payment rules. In other words, if agents best respond to the learned payment rules, do their strategies end up deviating greatly from truthful reporting?

It is also interesting to think about combining heuristic mechanism design with approximate incentive compatibility. In the work in Chapter 8 we perform sensitivity analysis and make BnB search fully monotone. This can be costly due to deallocations that occur when violations of monotonicity are discovered. We can imagine performing sensitivity analysis that decreases the number of monotonicity violations without making the outcome rule exactly monotone. There is a trade-off here between exact incentive compatibility and better welfare due to fewer deallocations.

Learning and mechanism design without money. Guo and Conitzer [2010a] investigate a mechanism design without money setting where artificial payments are used to construct a truthful in expectation mechanism for divisible items. The main idea is the following. Given a subset of items, design instantaneous pricing functions for each item (that do not depend on the agents' reports), and grant each agent a budget. To allocate the items, randomly order the agents and let agents spend their budget on the remaining items. Agents have no incentive to misreport since their reports do not affect the prices or budgets, and the agents will purchase the combination of items that maximizes their value. Guo and Conitzer [2010a] show how certain families of pricing functions can yield good worst case guarantees on social welfare (compared to the optimal social welfare).

An interesting direction for future work is to try and extend the learning framework from Chapter 9 to a mechanism design without money setting. Assuming the existence of a prior on agent types, perhaps machine learning methods can be used to learn pricing functions that have good expected social welfare (in contrast to the focus on worst-case analysis of Guo and Conitzer [2010a]). While the pricing functions studied by Guo and Conitzer [2010a] cannot depend on the agents' reports (otherwise an agent might misreport so that the other agents get higher or lower prices and purchase items that are more advantageous

to the agent), in a Bayesian setting it may be useful to split the agents and items into separate groups and use the reports of agents in other groups to price and allocate items among a group of agents. This brings the problem closer to the framework of Chapter 9 where payment rules can be functions of other agents' reports.

Selecting outcome rules based on regret. Adopting the results of Chapter 9 as a module, we can think about parameterizing a family of outcome rules and trying to find the parameterization that minimizes regret. It would be interesting to come up with some domains where we have natural parameterizations of outcome rules and adopt this extra layer on top of the framework for learning payment rules.

Bibliography

Ittai Abraham, Moshe Babaioff, Shaddin Dughmi, and Tim Roughgarden. Combinatorial auctions with restricted complements. In *Proceedings of the 13th ACM-EC Conference*, pages 3–16, 2012.

Itai Ashlagi, Mark Braverman, Avinatan Hassidim, and Dov Monderer. Monotonicity and implementability. *Econometrica*, 78(5):1749–1772, 2010.

Lawrence M. Ausubel and Paul Milgrom. The lovely but lonely Vickrey auction. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 1, pages 17–40. MIT Press, 2006.

Michael O. Ball, George L. Donohue, and Karla Hoffman. Auctions for the safe, efficient and equitable allocation of airspace system resources. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 20, pages 507–538. Cambridge University Press, 2006.

Xiaohui Bei and Zhiyi Huang. Bayesian incentive compatibility via fractional assignments. In *Proceedings of the 22nd SODA*, pages 720–733, 2011.

Michael Benisch, Norman Sadeh, and Tuomas Sandholm. A theory of expressiveness in mechanisms. In *Proceedings of the 23rd AAAI Conference*, pages 17–23, 2008.

Marcus Berliant, Karl Dunz, and William Thomson. On the fair division of a heterogeneous commodity. *Journal of Mathematical Economics*, 21:201–216, 1992.

Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997. ISBN 1886529191.

Martin Bichler, Andrew Davenport, Gail Hohner, and Jayant Kalagnanam. Industrial procurement auctions. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 23, pages 593–612. Cambridge University Press, 2006.

Sushil Bikhchandani, Shurojit Chatterji, Ron Lavi, Ahuva Mu'alem, Noam Nisan, and Arunava Sen. Weak monotonicity characterizes deterministic dominant-strategy implementation. *Econometrica*, 74(4):1109–1132, 2006.

Anna Bogomolnaia and Hervé Moulin. A new solution to the random assignment problem. *Journal of Economic Theory*, 100:295–328, 2001.

Anna Bogomolnaia and Hervé Moulin. Random matching under dichotomous preferences. *Econometrica*, 72:257–279, 2004.

Steven J. Brams and Alan D. Taylor. An envy-free cake division protocol. *The American Mathematical Monthly*, 102(1):9–18, 1995.

Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, 1996.

Steven J. Brams, Michael A. Jones, and Christian Klamler. Better ways to cut a cake. *Notices of the AMS*, 53(11):1314–1321, 2006.

Steven J. Brams, Michael A. Jones, and Christian Klamler. Proportional pie-cutting. *Int. Journal of Game Theory*, 36(3–4):353–367, 2008.

Steven J. Brams, Michal Feldman, John K. Lai, Jamie Morgenstern, and Ariel D. Procaccia. On maxsum fair cake divisions. In *Proceedings of the 26th AAAI Conference*, 2012a.

Steven J. Brams, Michael A. Jones, and Christian Klamler. N-person cake-cutting: There may be no perfect division. American Mathematical Monthly, forthcoming, 2012b.

Simina Brânzei and Peter Bro Miltersen. Equilibrium analysis in cake cutting. In *Proceedings of the 12th AAMAS Conference*, page forthcoming, 2013.

Simina Brânzei, Ariel D. Procaccia, and Jie Zhang. Externalities in cake cutting. In *Proceedings of the 23rd IJCAI*, page forthcoming, 2013.

Patrick Briest, Piotr Krysta, and Berthold Vöcking. Approximation techniques for utilitarian mechanism design. In *Proceedings of the 37th STOC*, pages 39–48, 2005.

Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. Working Paper, 2010.

Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. An algorithmic characterization of multi-dimensional mechanisms. In *Proceedings of the 44th STOC*, pages 459–478, 2012a.

Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. In *Proceedings of the 53rd FOCS Symposium*, pages 130–139, 2012b.

Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. Reducing revenue to welfare maximization: Approximation algorithms and other generalizations. In *Proceedings of the 24th SODA*, 2013. forthcoming.

Estelle Cantillon and Martin Pesendorfer. Auctioning bus routes: The london experience. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 22, pages 573–592. Cambridge University Press, 2006.

Chris Caplice and Yossi Sheffi. Combinatorial auctions for truckload transportation. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 21, pages 539–572. Cambridge University Press, 2006.

Ioannis Caragiannis, Christos Kaklamanis, Panagiotis Kanellopoulos, and Maria Kyropoulou. The efficiency of fair division. In *Proceedings of the 5th WINE*, pages 475–482, 2009.

Ioannis Caragiannis, John K. Lai, and Ariel D. Procaccia. Towards more expressive cake cutting. In *Proceedings of the 22nd IJCAI*, 2011.

Gabriel Carroll. A quantitative approach to incentives: Application to voting rules. Technical report, MIT, 2011.

Yiling Chen, John K. Lai, David C. Parkes, and Ariel D. Procaccia. Truth, justice, and cake cutting. In *Proceedings of the 24th AAAI Conference*, pages 756–761, 2010.

Yiling Chen, John K. Lai, David C. Parkes, and Ariel D. Procaccia. Truth, justice, and cake cutting. *Games and Economic Behavior*, 77(1):284–297, 2013.

Yann Chevaleyre, Paul E. Dunne, Ulle Endriss, Jérome Lang, Michel Lemaître, Nicolas Maudet, Julian Padget, Steve Phelps, Juan A. Rodríguez-Aguilar, and Paulo Sousa. Issues in multiagent resource allocation. *Informatica*, 30:3–31, 2006.

Yann Chevaleyre, Ulle Endriss, Sylvia Estivie, and Nicolas Maudet. Multiagent resource allocation with k-additive utility functions. *Annals of Operations Research*, 163(1):49–62, October 2008.

Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

E. Cohen, M. Feldman, A. Fiat, H. Kaplan, and S. Olonetsky. Truth, envy, and truthful market clearing bundle pricing. In *Proceedings of the 7th Workshop on Internet and Network Economics*, WINE '11, pages 97–108, 2011.

Yuga J. Cohler, John K. Lai, David C. Parkes, and Ariel D. Procaccia. Optimal envy-free cake cutting. In *AAAI*, 2011.

Vincent Conitzer and Tuomas Sandholm. Complexity of mechanism design. In *Proceedings of the 18th UAI Conference*, pages 103–110, 2002.

Vincent Conitzer and Tuomas Sandholm. Incremental mechanism design. In *Proceedings of the 20th IJCAI*, pages 1251–1256, 2007.

Vincent Conitzer, Tuomas Sandholm, and Paolo Santi. Combinatorial auctions with k-wise dependent valuations. In *Proceedings of the 20th AAAI Conference*, pages 248–254, 2005.

Florin Constantin and David C. Parkes. Self-correcting sampling-based dynamic multi-unit auctions. In *Proceedings of the 10th ACM-EC Conference*, pages 89–98, 2009.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

Peter Cramton. Spectrum auctions. In Martin Cave, Sumit Majumdar, and Ingo Vogelsang, editors, *Handbook of Telecommunications Economics*, chapter 14, pages 605–639. North-Holland/Elsevier, 2002.

Peter Cramton and Jesse A. Schwartz. Collusive bidding: Lessons from the fcc spectrum auctions. *Journal of Regulatory Economics*, 17:229 – 252, May 2000.

Peter Cramton, Yoav Shoham, and Richard Steinberg, editors. *Combinatorial Auctions*. Cambridge University Press, 2006.

Partha Dasgupta, Peter Hammond, and Eric Maskin. The implementation of social choice rules: some results on incentive compatibility. *Review of Economic Studies*, 46(2):185–216, 1979.

Robert Day and Paul Milgrom. Core-selecting package auctions. *International Journal of Game Theory*, 36(3–4):393–407, 2008.

Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *Proceedings of the 43rd FOCS Symposium*, pages 389–395, 2002.

L. E. Dubins and E. H. Spanier. How to cut a cake fairly. *American Mathematical Monthly*, 84(5):1–17, January 1961.

Shaddin Dughmi and Tim Roughgarden. Black-box randomized reductions in algorithmic mechanism design. In *Proceedings of the 51st FOCS Symposium*, pages 775–784, 2010.

Bhaskar Dutta and Debraj Ray. A concept of egalitarianism under participation constraints. *Econometrica*, 57:615–635, 1989.

Paul Dütting, Felix Fischer, and David C. Parkes. Simplicity-expressiveness tradeoffs in mechanism design. In *Proceedings of the 12th EC*, 2011.

Paul Dütting, Felix A. Fischer, Pichayut Jirapinyo, John K. Lai, Benjamin Lubin, and David C. Parkes. Payment rules through discriminant-based classifiers. In *Proceedings of the 12th EC*, pages 477–494, 2012.

Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, March 2007.

Jeff Edmonds and Kirk Pruhs. Cake cutting really is not a piece of cake. In *Proceedings of the 17th SODA*, pages 271–278, 2006a.

Jeff Edmonds and Kirk Pruhs. Balanced allocations of cake. In *Proceedings of the 47th FOCS Symposium*, pages 623–634, 2006b.

Edmund Eisenberg and David Gale. Consensus of subjective probabilities: The pari-mutuel method. *The Annals of Mathematical Statistics*, 30(1):165–168, 1959.

Aytek Erdil and Paul Klemperer. A new payment rule for core-selecting package auctions. *Journal of the European Economic Association*, 8(2–3):537–547, 2010.

S. Even and A. Paz. A note on cake-cutting. *Discrete Applied Mathematics*, 7:285–296, 1984.

Paul Feautrier. Parametric integer programming. *RAIRO Recherche Op'erationnelle*, 22, 1988.

Michal Feldman and John K. Lai. Mechanisms and impossibilities for truthful, envy-free allocations. In *SAGT*, pages 120–131, 2012.

Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the 16th IJCAI*, pages 548–553, 1999.

Allan Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

Jerry Green and Jean-Jacques Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45(2):427–438, 1973.

Theodore Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

M. Guo and V. Conitzer. Strategy-proof allocation of multiple items without payments or priors. In *Proceedings of the 9th AAMAS Conference*, pages 881 – 888, 2010a.

Mingyu Guo and Vincent Conitzer. Computationally feasible automated mechanism design: General approach and case studies. In *Proceedings of the 24th AAAI Conference*, 2010b.

Jason D. Hartline and Anna R. Karlin. Profit maximization in mechanism design. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 13, pages 331–361. Cambridge University Press, 2007.

Jason D. Hartline and Brendan Lucier. Bayesian algorithmic mechanism design. In *Proceedings of the 42nd STOC*, pages 301–310, 2010.

Jason D. Hartline, Robert Kleinberg, and Azarakhsh Malekian. Bayesian incentive compatibility via matchings. In *Proceedings of the 22nd SODA*, pages 734–747, 2011.

Bengt Holmström. Groves schemes on restricted domains. *Econometrica*, 47(5):1137 – 1144, 1979.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 186–202, 2010.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, 2009.

Akshay-Kumar Katta and Jay Sethuraman. A solution to the random assignment problem on the full preference domain. *Journal of Economic Theory*, 131:231–250, 2006.

David Kurokawa, John K. Lai, and Ariel D. Procaccia. How to cut a cake before the party ends. In *Proceedings of the 27th AAAI Conference*, 2013. forthcoming.

Sébastien Lahaie. A kernel method for market clearing. In *Proceedings of the 21st IJCAI*, pages 208–213, 2009.

Sébastien Lahaie. Stability and incentive compatibility in a kernel-based combinatorial auction. In *Proceedings of the 24th AAAI Conference*, pages 811–816, 2010.

Sébastien Lahaie. A kernel-based iterative combinatorial auction. In *Proceedings of the 25th AAAI Conference*, pages 695–700, 2011.

John K. Lai and David C. Parkes. Monotone branch-and-bound search for restricted combinatorial auctions. In *Proceedings of the 12th EC*, pages 705–722, 2012.

Ron Lavi and Chaitanya Swamy. Truthful and near-optimal mechanism design via linear programming. *J. ACM*, 58(6):25, 2011.

Ron Lavi, Ahuva Mu'alem, and Noam Nisan. Towards a characterization of truthful combinatorial auctions. In *Proceedings of the 44th FOCS Symposium*, pages 574–583, 2003.

Daniel Lehmann, Liadan I. O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, 2002.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM-EC Conference*, pages 66–76, 2000.

Anton Likhodedov and Tuomas Sandholm. Approximating revenue-maximizing combinatorial auctions. In *Proceedings of the 20th AAAI Conference*, pages 267–274, 2005.

Richard Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 6th EC*, pages 125–131, 2004.

Benjamin Lubin. *Combinatorial Markets in Theory and Practice: Mitigating Incentives and Facilitating Elicitation.* PhD thesis, School of Engineering and Applied Sciences, Harvard University, 2010.

Benjamin Lubin and David C. Parkes. Quantifying the strategyproofness of mechanisms via metrics on payoff distributions. In *Proceedings of the 25th UAI Conference*, pages 349–358, 2009.

Benjamin Lubin and David C. Parkes. Approximate strategyproofness. *Current Science*, 103(9):1021–1032, 2012.

Roy E. Marsten and Thomas Morin. Parametric integer programming: The Right-Hand side case. *Annals of Discrete Mathematics*, 1:375–390, 1977.

Andreau Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

Avishay Maya and Noam Nisan. Incentive compatible two player cake cutting. In *Proceedings of the 8th WINE*, pages 170–183, 2012.

Nimrod Megiddo. A good algorithm for lexicographically optimal flows in multi-terminal networks. *Bulletin of the American Mathematical Society*, 83:407–409, 1979.

Elchanan Mossel and Omer Tamuz. Truthful fair division. In *Proceedings of the 3rd SAGT*, 2010. To appear.

Ahuva Mu'alem and Noam Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. *Games and Economics Behavior*, 64:612–631, 2008.

Roger B. Myerson. Incentive-compatibility and the bargaining problem. *Econometrica*, 47 (1):61–73, 1979.

Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 1:58–73, 1981.

George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. J. Wiley and Sons, Inc., 1998.

Jerzy Neyman. Un théorèm d'existence. *C. R. Acad. Sci. Paris*, 222:843–845, 1946.

Noam Nisan. Bidding languages for combinatorial auctions. In P. Cramton, Y. Shoham, and P. Steinberg, editors, *Combinatorial Auctions*, chapter 9, pages 215–232. MIT Press, 2006.

Noam Nisan. Introduction to mechanism design (for computer scientists). In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 9. Cambridge University Press, 2007.

Noam Nisan. Google's auction for tv ads. In *SODA*, page 741, 2010.

Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

Stephen W. Nuchia and Sandip Sen. Improving optimality of n agent envy-free divisions. In *Proceedings of the 8th ATAL*, pages 277–289, 2001.

Christos H. Papadimitriou. Efficient search for rationals. *Information Processing Letters*, 8:1–4, 1979.

S. Pápai. Groves sealed bid auctions of heterogeneous objects with fair prices. *Social Choice and Welfare*, 20(3):371 – 385, 2003.

David C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, 2001.

David C. Parkes. When analysis fails: Heuristic mechanism design via self-correcting procedures. In *Proceedings of the 35th SOFSEM*, pages 62–66, 2009.

David C. Parkes and Quang Duong. An ironing-based approach to adaptive online mechanism design in single-valued domains. In *Proceedings of the 22nd AAAI Conference*, pages 94–101, 2007.

David C. Parkes and Grant Schoenebeck. Growrange: Anytime vcg-based mechanisms. In *Proceedings of the 19th AAAI Conference*, pages 34–41, 2004.

David C. Parkes, Jayant Kalagnanam, and Marta Eso. Achieving budget-balance with Vickrey-based payment schemes in exchanges. In *Proceedings of the 17th IJCAI*, pages 1161–1168, 2001.

Parag Pathak and Tayfun Sönmez. Comparing mechanisms by their vulnerability to manipulation. Working Paper, 2010.

Ariel D. Procaccia. Thou shalt covet thy neighbor's cake. In *Proceedings of the 21st IJCAI*, pages 239–244, 2009.

Ariel D. Procaccia. Cake cutting: Not just child's play. *Communications of the ACM*, 2013. forthcoming.

Baharak Rastegari, Anne Condon, and Kevin Leyton-Brown. Revenue monotonicity in deterministic, dominant-strategy combinatorial auctions. *Artificial Intelligence*, 175:441–456, 2011.

J.H. Reijnierse and J. A. M. Potters. On finding an envy-free Pareto-optimal division. *Mathematical Programming*, 83:291–311, 1998.

Jack Robertson and William Webb. *Cake Cutting Algorithms: Be Fair If You Can*. A. K. Peters, 1998.

Michael H. Rothkopf, Aleksandar Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

Amin Saberi and Ying Wang. Cutting a cake for five people. In *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management*, pages 292–300, 2009.

Michael Saks and Lan Yu. Weak monotonicity suffices for truthfulness on convex domains. In *Proceedings of the 6th ACM-EC Conference*, pages 286–293, 2005.

Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

Tuomas Sandholm. Optimal winner determination algorithms. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 14, pages 337–368. MIT Press, 2006.

Tuomas Sandholm. *Very-Large-Scale Generalized Combinatorial Multi-Attribute Auctions: Lessons from Conducting $60 Billion of Sourcing*. Oxford University Press, 2013. To appear.

Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Cabob: A fast optimal algorithm for winner determination in combinatorial auctions. *Management Science*, 51 (3):374–390, 2005.

James Schummer. Strategy-proofness versus efficiency on restricted domains ofexchange economies. *Social Choice and Welfare*, 14(1):47–56, 1997.

Hugo Steinhaus. The problem of fair division. *Econometrica*, 16:101–104, 1948.

Francis E. Su. Rental harmony: Sperner's lemma in fair division. *American Mathematical Monthly*, 106(10):930–942, 1999.

Benjamin Taskar, Vassil Chatalbashev, and Daphne Koller. Learning associative markov networks. In *Proceedings of the 21st ICML*, 2004.

William Thomson. Children crying at birthday parties. Why? *Journal of Economic Theory*, 31:501–521, 2007.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Vijay V. Vazirani. Combinatorial algorithms for market equilibria. In N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 5. Cambridge University Press, 2007.

William Vickrey. Counter speculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

Roie Zivan, Miroslav Dudík, Steven Okamoto, and Katia P. Sycara. Reducing untruthful manipulation in envy-free Pareto optimal resource allocation. In *Proceedings of the 10th IAT*, pages 391–398, 2010.